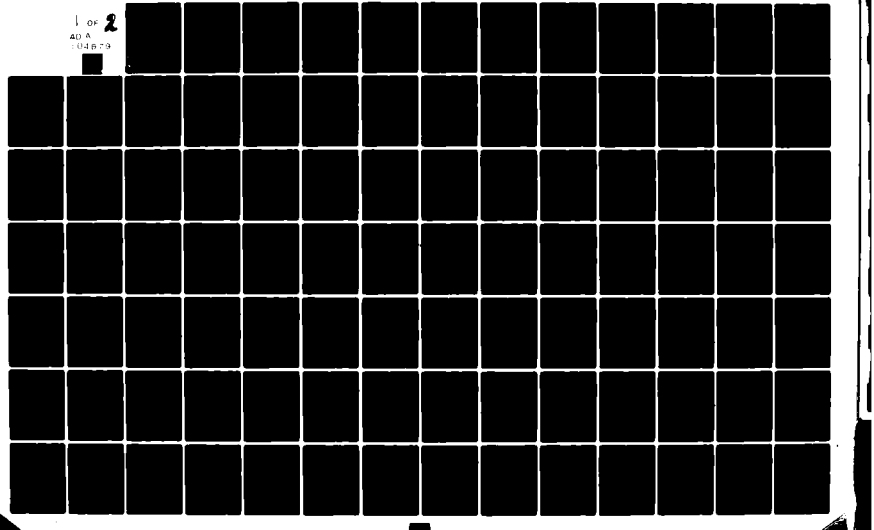
AD-A104 679 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH F/G 9/2
FIDELITY OPTIMIZATION OF MICROPROCESSOR SYSTEM SIMULATIONS.(U)
MAR 81 E T LANDRUM
UNCLASSIFIED AFIT-CI-81-3T NL



UNCLASS
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

LEVEL

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

REPORT NUMBER	2 GOVT ACCESSION NO	3 RECIPIENT'S CATALOG NUMBER
81-3T	AD-A104 679	

4. TITLE (and Subtitle)
Fidelity Optimization of Microprocessor System Simulations.

5 TYPE OF REPORT & PERIOD COVERED
THESIS/DISSERTATION

6 PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
Earnest Taylor Landrum, Jr.

8. CONTRACT OR GRANT NUMBER(s)

9 PERFORMING ORGANIZATION NAME AND ADDRESS
AFIT STUDENT AT: Auburn University

10 PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
AFIT/NR
WPAFB OH 45433

12. REPORT DATE
March 1981

13. NUMBER OF PAGES
91

14 MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)

15. SECURITY CLASS. (of this report)
UNCLASS

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

1981

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES
APPROVED FOR PUBLIC RELEASE: IAW AFR 190-17

Fredric C. Lynch
FREDRIC C. LYNCH, Major, USAF
Director of Public Affairs
Air Force Institute of Technology (ATC)
Wright-Patterson AFB, OH 45433

23 JUN 1981

19 KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
ATTACHED

81 7 16 025

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASS
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AFIT RESEARCH ASSESSMENT

81-3T

The purpose of this questionnaire is to ascertain the value and/or contribution of research accomplished by students or faculty of the Air Force Institute of Technology (ATC). It would be greatly appreciated if you would complete the following questionnaire and return it to:

AFIT/NR
Wright-Patterson AFB OH 45433

RESEARCH TITLE: Fidelity Optimization of Microprocessor System Simulations

AUTHOR: Earnest Taylor Landrum, Jr.

RESEARCH ASSESSMENT QUESTIONS:

1. Did this research contribute to a current Air Force project?

( ) a. YES ( ) b. NO

2. Do you believe this research topic is significant enough that it would have been researched (or contracted) by your organization or another agency if AFIT had not?

( ) a. YES ( ) b. NO

3. The benefits of AFIT research can often be expressed by the equivalent value that your agency achieved/received by virtue of AFIT performing the research. Can you estimate what this research would have cost if it had been accomplished under contract or if it had been done in-house in terms of manpower and/or dollars?

( ) a. MAN-YEARS _____ ( ) b. $_____

4. Often it is not possible to attach equivalent dollar values to research, although the results of the research may, in fact, be important. Whether or not you were able to establish an equivalent value for this research (3. above), what is your estimate of its significance?

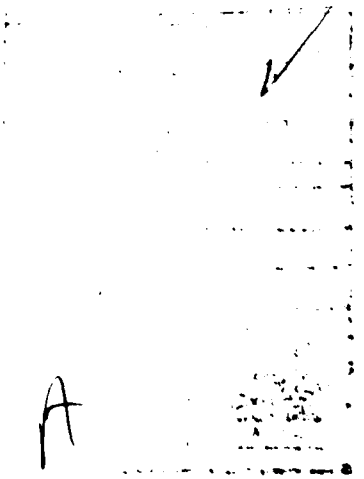
( ) a. HIGHLY SIGNIFICANT ( ) b. SIGNIFICANT ( ) c. SLIGHTLY SIGNIFICANT ( ) d. OF NO SIGNIFICANCE

5. AFIT welcomes any further comments you may have on the above questions, or any additional details concerning the current application, future potential, or other value of this research. Please use the bottom part of this questionnaire for your statement(s).

NAME GRADE POSITION

ORGANIZATION LOCATION

STATEMENT(s):



FOLD DOWN ON OUTSIDE - SEAL WITH TAPE

AFIT/NR
WRIGHT-PATTERSON AFB OH 45433

OFFICIAL BUSINESS
PENALTY FOR PRIVATE USE. $300

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 73236 WASHINGTON D.C.

POSTAGE WILL BE PAID BY ADDRESSEE

AFIT/ DAA
Wright-Patterson AFB OH 45433

FOLD IN

FIDELITY OPTIMIZATION OF MICROPROCESSOR SYSTEM SIMULATIONS

Earnest Taylor Landrum, Jr.

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Master of Science

Auburn, Alabama

March 19, 1981

FIDELITY OPTIMIZATION OF MICROPROCESSOR

SYSTEM SIMULATIONS

Earnest Taylor Landrum, Jr.

Certificate of Approval:

V. P. Nelson, Assistant Professor Chairman Electrical Engineering	J. D. Irwin, Professor Electrical Engineering
J. S. Boland, Professor Electrical Engineering	Paul F. Parks, Dean Graduate School

FIDELITY OPTIMIZATION OF MICROPROCESSOR

SYSTEM SIMULATIONS

Earnest Taylor Landrum, Jr.

Earnest T Landrum Jr.
Signature of Author

3 February 1981
Date

Copy sent to:

Name Date

VITA

Earnest Taylor Landrum, Jr., son of Earnest T. and Lois (Dean) Landrum, was born July 24, 1948, in San Antonio, Texas. He attended Greenville County, South Carolina, public schools and graduated from Greenville Senior High School, Greenville, South Carolina, in 1966. In September 1966 he entered the Georgia Institute of Technology and received the degree of Bachelor of Science (Physics) in June 1970. He then entered the United States Air Force as a second lieutenant. He entered graduate studies at Auburn University in June 1977. He married Kathleen, daughter of Edwin J. and Ethel (Hoeck) Clisham in June 1977. They have one daughter, Jessica Dean.

THESIS ABSTRACT

FIDELITY OPTIMIZATION OF MICROPROCESSOR

SYSTEM SIMULATIONS

Earnest Taylor Landrum, Jr.

Master of Science, March 19, 1981
(B.S., Georgia Institute of Technology, 1970)

117 Typed Pages

Directed by Victor P. Nelson

The development of a microprocessor system simulation that would accurately portray the operation of the system at a very fine level of detail was studied. This optimization in the area of fidelity was broken into three tasks. A preprocessor program was written to improve the operator interface to an existing simulation driver program. An existing microprocessor simulation, designed to run under the simulation driver program, was extensively modified to reflect actual machine level operations rather than abstract level functions. A simulation of a programmable parallel interface was developed and mated to the microprocessor simulation. Examples and possibilities for system level simulation are discussed and analyzed.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

The work to be described in this thesis focuses on simulation of microprocessor-based systems, pursuing three related objectives. First, simulation routines must have an efficient human interface to allow effective interaction with the user and efficient use of the simulation capabilities. Secondly, the simulation program of the target machine should be a highly faithful model of that machine, to allow use of the simulation results with a minimum of corrections for simulator-based peculiarities. Finally, the target machine must be complete enough to accurately portray system operation, including input/output functions.

Thus, there were three logically connected tasks to be done. The first task was to develop a preprocessor program to increase the utility and ease of operation of an existing simulator program, based on a hardware description language. The second task was to develop a microprocessor simulation, avoiding the abstract level in favor of one more in line with the actual operation of the target machine. The final task was to develop the capability to simulate a complete system with input/output functions.

The simulator program used was the Computer Design Language Simulator - USF Version 2, as run on the computer system of Auburn University. This simulator program is based on Computer Design Language (CDL), a hardware description language developed by Dr. Yoahan Chu of

the University of Maryland (1,2). Using an algebraic structure, CDL describes device operations at the register transfer level. The main advantage of the language is this logical struccure. Hardware devices are called by commonly used names and register transfer operations are easily understood. The simulator program retains this clear, logical translation of a hardware system into CDL. However, there are two disadvantages to using the program. Initial program and data load of the target machine must be prepared in binary machine code, which can be awkward. In addition, considerable amount of processing time is necessary, due to the intensely iterative nature of the simulation routine.

The preprocessor developed was designed to remedy one of these drawbacks. The preprocessor allows the use of assembly language to load the target machine's simulated program space. This human interface frees the user to concentrate on the results of the simulation rather than on the mechanics of achieving it. It also provides a simple set of format and semantic checks to be made on the program to be assembled. The prime requirement was to make simulation easier to achieve and correct, thus more responsive to the user.

A simulation of a microprocessor was available as a result of an earlier study (3). However, many of the routines were written only to provide a correct output, without regard to the mechanism used. The simulation was extensively modified to more closely duplicate the actual operation of the target machine. The functions of the basic support chips were defined more explicitly. Using the improved simula-

tion as a basis, the functions of representative communications chips were developed. The unique aspect of the resulting product was its ability to model an integrated system, including input/output and interrupt driven routines.

The body of this paper will further describe these three tasks. The considerations and constraints used in the development of the preprocessor are described first. The next section discusses both the principles used to modify the Intel 8080 simulation for increased fidelity and those principles used to build a parallel communications interface. The experimental results obtained from testing the system are then presented. The final section contains the conclusions drawn from the project and some suggested directions for further work in this area.

II. PREPROCESSOR DEVELOPMENT

The CDL Simulator Program is designed for hardware simulation at the register transfer level. At this level, a processor operates by logically decoding commands and data presented in machine language. CDL is particularly efficient in expressing the decoding and execution processes. Although this feature offers great flexibility and detail in design, it becomes a drawback when simulating the execution of trail programs, due to the necessity of translating these programs into machine language. The preprocessor's major function is to translate programs written in the assembly language of the target machine into CDL-compatible machine code and load them into the assigned memory space. It operates as an assembler and loader, with appropriate support functions such as symbol table generation. The output of the assembler routine is presented in two forms for user convenience. The first version is a line by line translation of the assembly code. The program is displayed for analysis and correction of errors. The second version is the CDL-compatible card image, displaying the machine code as it is presented to the simulation program. This version is particularly helpful in tracing the execution of the simulation.

The second design goal was to provide the translation process with an adequate human interface. Careful design of the output, as discussed above, was a first step. Although the preprocessor was never intended to be a complete software development tool, routines were in-

cluded to detect and flag the type of errors likely to be made in executing trial programs on a simulated machine. These include both syntax errors in the structure of the trial program and coding errors within the program itself. These routines are limited to those that would be most useful.

An additional constraint was imposed on the preprocessor. To be compatible with the existing CDL simulation program, it has to be written in FORTRAN. FORTRAN, however, lacks the bit-level instructions necessary to deal with character data. Following the example of the basic simulation program, the preprocessor implements several required functions in IBM 370 assembly language subroutines. While there is a bonus in increased execution speed, program linkage and integration posed significant problems during development.

The choice of a target machine was also an important consideration. Since the preprocessor works with assembly language, a target machine had to be chosen in order to code the assembler. For maximum utility, the preprocessor would have to work with a significant machine, one having widespread use and a need to be simulated. It would also have to be one that had information on its internal operation widely available. The choice for this work was the Intel 8080 microprocessor.

The basic function of the preprocessor is that of a standard two pass assembler and loader (4). While FORTRAN does not lend itself to the writing of structured programs, an attempt was made to preserve logical form in the program (Figure 1). The program has a central FORTRAN driver routine, ASMINT, that performs initialization, selects

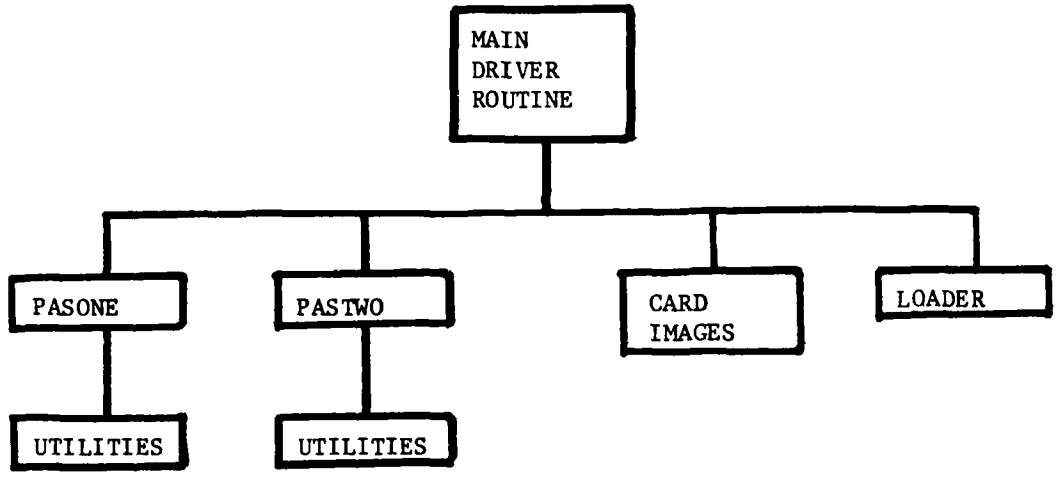


Figure 1. Preprocessor Program Structure

the required language set, directs the passes of the assembler and outputs the final code, both as hard copy and CDL compatible machine code in the program memory space. These major functions are implemented in FORTRAN subroutines, in turn supported as necessary by IBM 370 assembly language subroutines (Table 1). Assembly language is used in routines performing bit manipulation and in the routines that are highly iterative. This structure supports the design objectives of the preprocessor. As part of a time-consuming, intensely iterative program, this segment has to be relatively fast to avoid lengthening an already long program in execution. To conserve memory space and improve speed, it has to be relatively small. To improve readability and encourage both use and future improvements, it has to be relatively straightforward and simple. The overall design strategy was to produce a limited implementation that stressed utility over optimization. The prominent features of the preprocessor are listed in Table 2. For a more complete description of the features and options of the program, both the user's manual and program listing are included in this work as appendices.

Integration of the preprocessor into the CDL simulation program posed several problems. There was the language problem described earlier with the mating of FORTRAN and IBM 370 assembly language. The preprocessor also had to integrate with the CDL simulator in such a way as to preserve the human interface and the logical continuity of the main program. The design solution to this problem was to have the preprocessor produce card images of the assembled machine code and load them in accordance with the procedures for loading CDL simulator

Level	Title	Language	Function
First	ASMINT	FORTRAN	Assembler driver routine
Second	PASONE	FORTRAN	Assembler first pass driver
	PASTWO	FORTRAN	Assembler second pass driver
	IMAGER	FORTRAN	Build card image format
Third	Utilities		Character manipulation and assembly
	POPSUB	FORTRAN	Pseudo-op handling
	LODASM	Assembly	Operation code table loader
	LABLST	Assembly	Symbol table manager
	PCODE	Assembly	Operation code table manager
	STRING	Assembly	String decoder
	OPERAN	Assembly	Operand numerical converter
	VALRED	Assembly	Character numerical converter

Table 1. Preprocessor Routine List

Assembler options

- Language
- Symbol table listing
- Location counter initialization

Data types

- Numerical (decimal,hexadecimal, octal, binary)
- Character strings
- Expressions

Pseudo-ops

- Assembler control (origin and end)
- Data storage (byte, word, space, equality)

Input — Assembly language program

Output

- Assembled code listing
- Loader compatible card images

Table 2. Prominent Preprocessor Features

memory space from cards. This approach maintained the continuity of the CDL simulator and relieved the necessity of creating an alternate method of introducing data into the assigned program memory space of the simulator.

Even though a specific target machine was chosen, the preprocessor was designed to permit extension into other languages to make it more versatile. One of the initial operator specifications is the language to be used by the assembler. This specification controls the operation code set selected by the program. The pseudo operation codes are indexed to allow multiple routines to be written to accommodate the different languages. Complete commonality, of course, is impossible to achieve. The IBM 370 assembly language subroutines for handling operands and addressing were specifically written to generate Intel 8080 code. However, the modular structure of the preprocessor would allow them to be replaced with subroutines suited for the desired language.

III. FIDELITY OPTIMIZATION OF A SIMULATION

One of the most important attributes of a hardware simulator is fidelity, the degree to which the simulation approximates reality. Optimization of fidelity is the process of balancing the requirements of broad principles of simulation, alternative methods of representation available in specific cases, and the priorities in performance factors of the simulation as a whole. The desired outcome is a faithful simulation that sacrifices as little as possible in attaining fidelity. This chapter describes the optimization process as applied to the specific case of the Intel 8080 microprocessor within the constraints of the CDL simulator.

Microprocessor Simulation

The operation of the Intel 8080 CPU can be analyzed down to a fine level. An instruction cycle is the time it takes to fetch and execute a single instruction. A machine cycle is generated each time a memory or I/O access is made. This machine cycle can be subdivided into separate states. In these individual states the actual microoperations of the CPU take place. Depending on the number and type of microoperations executed within the machine cycle, there are three, four, or five states in that cycle. The number of machine cycles required to complete an instruction depends upon the number of accesses to memory or I/O. All of the 8080 instructions can be broken down in

terms of machine cycles and states (5). This analysis forms the basis for the reality that must be simulated.

There are, however, areas of operation where certain assumptions must be made to accommodate the hardware description language to the processor. An outstanding example in this project is the use of flag registers. The exact hardware logic used within the microprocessor to initiate certain sequences is embedded in the control circuits designed by the manufacturer. In order to allow the simulated microprocessor to initiate these sequences, nonexistent hardware registers have to be defined and assigned these functions. The prime example in the instruction execution portion of the simulation is the register labeled MREF. This flag is set whenever an instruction is to be executed using a memory reference as an assigned register operand. This register may not exist in the actual hardware or may not be accessible by the user. However, the simulator program can read the status of this flag register and use the results to implement the sequence of register transfers implemented in reality. The result is increased fidelity of operation. Further use of this technique is made in the implementation of control logic and will be discussed more fully in a later section.

Other general concepts should be considered within an improved simulation. The size of the simulation must be kept to a minimum by avoiding duplicate procedures. Transfer of control between similar operations is used where practical to achieve this goal. In a similar vein, the concept of execution overlap requires special handling. The 8080 microprocessor uses an overlap of the final processes of certain

instructions and the fetch of the next instruction. This overlap is used to increase the execution speed of the machine. CDL can directly support concurrent processes only in certain cases. Inclusion of the required extra routines to achieve the overlap is not justified by the small return in authenticity. The originally overlapped processes are generally included in the last scheduled machine cycle of the instruction in this simulation. The execution speed increase is thus preserved by performing the processes outside of machine time and the process is transparent, except at the precise moment of the overlap. In a few cases the simulation could not perform the required functions in the required time, even though they were not overlapped. In such instances, the simulation was designed to come as close as possible. These instances simply represent the limits of the ability of the simulation, normally visible only at the subcycle level.

The process of bringing a simulation into strict compliance with the actual operating principles is best done in several stages. A program had been developed to simulate the Intel 8080 in a multiprocessing environment (3). Therefore, the program was concerned primarily with the results of program execution and the transfer of control rather than the strict simulation of a microprocessing system. It is the basis for the instruction execution routine portion of the improved simulation. Varying degrees of fidelity required varying approaches. Some routines were completely rewritten. The HLT instruction is one example (Figure 2). The original sequence simply disables the software mechanism used to translate clock pulses into increasing machine cycle numbers. The revised sequence recognizes

ORIGINAL VERSION

```
Hlt

/M(1)*T(4)*P(1)*READY*IR(7)'*IR(6)/ IF (OP1(3)*OP2(2)*OP3(6)) THEN
          (READY=0, X=0, Y=1) ELSE (DO/SEVAL)
```

EXPANDED VERSION

```
Hlt

/M(1)*T(1)*P(1)*READY*IR(7)'*IR(6)/ IF (OP1(3)*OP2(2)*OP3(6)) THEN
          (HLTA=1, X=0, Y=2) ELSE (DO/SEVAL)

/M(2)*T(1)*P(1)*HLTA*READY/ SYNC=1, MEMR=1

/M(2)*T(2)*P(1)*HLTA/ WAIT=1, READY=0
```

Figure 2. HLT Command Routine

the halt, broadcasts it as the system status and enters a wait state before disabling the software driver. The additional actions are necessary to enable the processor to communicate with other parts of a complete system.

Some instructions were changed to make more efficient use of the memory and I/O routines developed in the control sections. The STAX and LDAX execution routine was expanded to include the memory cycle that occurs and makes the instruction continue into a second machine cycle. Several instructions were thus modified to show single byte I/O transfer. Adding a cycle was generally done in a straightforward manner. The single exception was the immediate instruction handler (Figure 3). This routine recognizes the immediate instruction type and fetches the required operand, using an added memory cycle. At this point, the machine cycle numbers being carried by the simulation are incorrect, even though the elapsed timing is very close to the actual. However, the only alternative is to reproduce all of the affected instruction execution routines, changing only the machine cycle numbers, and then add them to the instruction set. The option that was chosen was to maintain the smaller set of routines and accept the single exception rather than to pay the simulation execution speed penalty for redundant code. The simulation that results from the sum of all these actions is a quite accurate model of the Intel 8080 instruction set. The next section will discuss the development of the corresponding control logic.

ORIGINAL VERSION

Immediate Instruction Handler

```
/M(1)*T(4)*P(1)*READY*(IR(7).ERA.IR(6))'*OP3(6)/ ADDBUFFER=PC, SYNC=1,
          NWR=1, DBIN=1, WAIT=1, READY=0

/M(1)*T(4)*P(1)*READY*(IR(7).ERA.IR(6))'*OP3(6)/ PC=ADDBUFFER.COUNT.,
          TEMP=DATABUFF, IR(6)=IR(6)', X=4
```

EXPANDED VERSION

Immediate Instruction Handler

```
/M(1)*T(4)*P(1)*READY*(IR(7).ERA.IR(6))'*OP3(6)/ ALATCH=PC, MR1=1,
          X=0, Y=2

/M(2)*T(2)*P(1)*READY*(IR(7).ERA.IR(6))'*OP3(6)/ MR1=0

/M(2)*T(3)*P(1)*READY*(IR(7).ERA.IR(6))'*OP3(6)/ PC=ALATCH,
          TEMP=DATABUFF, IR(6)=IR(6)', X=4, Y=1
```

Figure 3. Immediate Instruction Handler Routine

Control Functions

The most important feature of faithful simulation of a microprocessor system is control function implementation. While instruction set implementation is easily structured to conform to the actual CPU microoperation sequences, the control sequences are the key to system level simulation. The control sequences must operate on two levels. The first level is basic system control of the CPU and associated support modules. Functions at this level include generation of CPU status information and basic memory access. The second level of control functions are those necessary to drive unique system modules. A specific example of this level is a communications module used to communicate with a system peripheral.

CPU Support Group

The first level of control applies to the Intel 8080 CPU support group of modules (Figure 4). This group includes the 8080 8-bit Microprocessor, the 8224 Clock Generator and Driver, and the 8228 System Controller and Bus Driver (6). The CPU itself has few control functions that are solely internal. One example is the clock cycle incrementor function which translates the incoming clock pulses into correct machine cycle and state signals. In the simulation this mechanism also implements the asynchronous interrupt function. The other control signals involve associated modules. The 8224 module is actually not separately simulated. Its clock functions are implicit in the two phase clock defined in the hardware section. Its only other function, converting the CPU synchronization signal to a

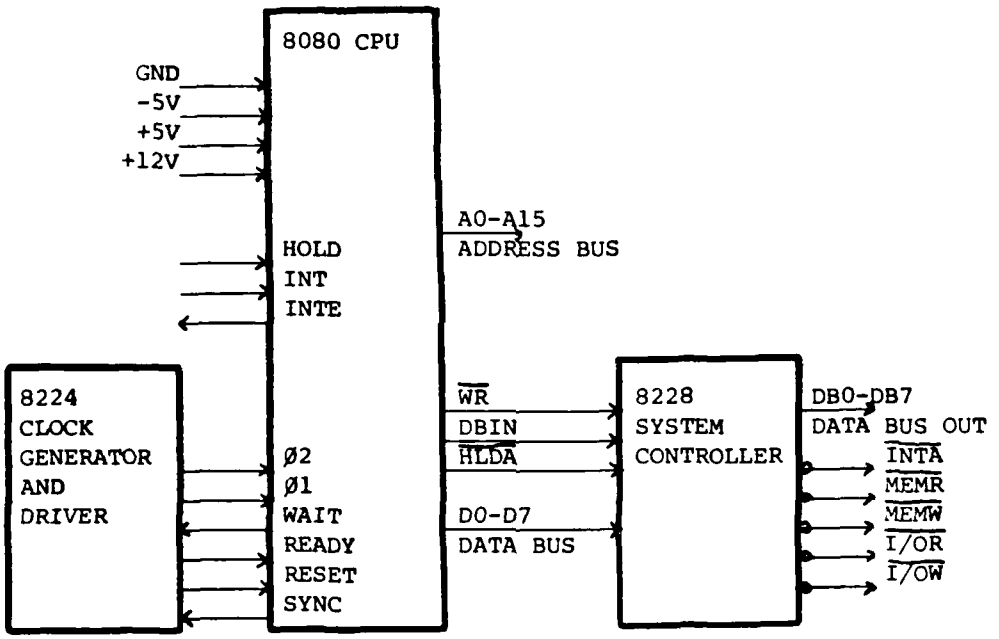


Figure 4. CPU Group

status strobe signal, is simulated separately. The bulk of the control signals are concentrated in the 8228. Its major function is the broadcast and application of the system status. Triggered by the status strobe, the simulation of the 8228 latches the status word from the data bus and combines elements of that status word and signals from the CPU in a gating array to generate memory and I/O access signals.

The control function simulations are designed to operate similar to nested subroutines. The normal memory routines activate selected status word registers and the synchronization pulse. The synchronization pulse triggers the status strobe, which loads the 8228 status latch via the data bus. The gating array activates the primary sequences for memory access, listed in the simulation as utility routines, and deactivates the ready signal, stopping the software driven cycle clock. After the services are performed, the ready signal is reactivated to allow the clock cycle incrementor to continue. This action simulates the access speed requirements. In this particular application, the memory is assumed to be sufficiently fast that the wait state need not be entered during the access. The machine cycle and state numbers remain correct. However, since CDL requires that all actions be driven by the system clock, the two clock cycles needed to complete the access are counted. This fact affects any timing analysis interpretation.

I/O Support

The module simulated for the second layer of the system is the Intel 8255 Programmable Peripheral Interface (7). This device uses a system software generated control word to program the functional characteristics of three eight bit ports to achieve a great number of input and output configurations. The 8255 was chosen for its versatility in controlling parallel communication. The programmable configuration feature makes it a very flexible device. However, the CDL simulator driver cannot support an ambiguously defined architecture. The hardware definition section accepts only a single description of each part of the system. Once the system design has been translated by the simulation driver, that design remains fixed throughout the simulation run, restricting the utility of the software driven functional control. Complete flexibility could be obtained only by including software instructions for every possible configuration, down to a single bit level. These instructions would have to be evaluated on each iteration of the simulator to construct the correct interface. The software overhead penalty of processing all the instructions for the other unused configurations was considered excessive. Therefore a single representative configuration was chosen for simulation.

The chosen configuration contains one strobed bi-directional bus and one input port, both with appropriate handshaking control lines (Figure 5). Full interrupt and strobing capabilities are included in the simulated logic. While software control of the configuration is not possible, the set/reset function of the control lines is implemented to allow control of the handshaking signals. Appropriate chip

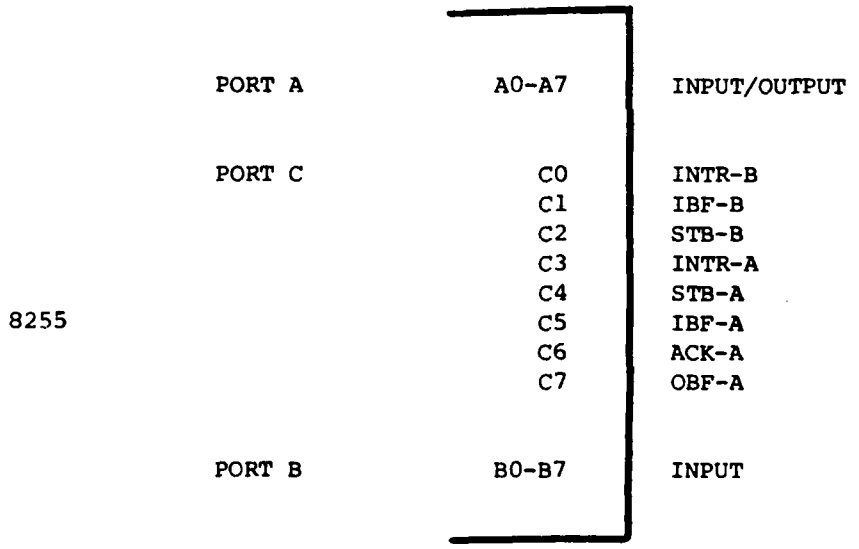


Figure 5. Simulated Intel 8255 Configuration

select and port select decoding logic is also simulated. Two routines are added for simplicity in the simulation. The first is an initialization routine which establishes the starting states of the handshaking lines. This routine shortens the simulated program by removing some housekeeping sequences. The second routine corresponds to a switch setting that simulates input to the 8255 by transferring memory data to the input port. This routine was necessary since there is no way to input external data to a CDL simulated machine during a simulation run. Switch statements, which are internal to the program, can only simulate true external inputs. Any process must be self-contained, as is this one.

Although parallel communication is a fairly straightforward register transfer operation, simulation of serial communication in CDL is a more complex task. The actual hardware simulation is relatively simple. Necessary components would include a holding register for the byte being transferred, a pointer to the next bit to be handled, and logic to implement the necessary line protocol. The complexity arises in timing the transfer of the information. As noted before, the CDL simulator does not handle concurrent tasking well. A central clock is defined which provides all timing information. As in the CPU simulation, a software counter mechanism would be necessary to define the internal timing for the serial transfer. The hardware and software constructs necessary for the serial interface would lengthen the simulation substantially. The simulation driver has a limit on the amount of hardware that it can incorporate into the translated architecture. There is no limit imposed on the software, but due to the sequential,

iterative nature of the driver, a penalty in execution speed is paid for each statement. These limitations did not favor an additional interface.

A second reason for not implementing the serial interface was evident from the nature of its operation. For low speed applications, at 300 bits per second, the software timing counter would need a maximum count of 6,600 central clock cycles to process a single bit. High speed applications, typically 2400 bits per second, would still require around 830 cycles per bit. Based on simulation runs made in this project, such a simulation would require in excess of ten minutes of CPU time for that single bit transfer at 2400 bits per second, due to the granularity of the time base. The time could be reduced by simulating the serial interface separately from the rest of the system. Accomplishing this simulation would require that the hardware design described earlier and the appropriate logic functions for that design be substituted for the sections relating to the 8255 module. This would allow serial communication simulation, but not parallel communication simulation. If the two were to be simulated together, one way to make the effort feasible in terms of required CPU time would be to employ a separate clock with an artificially compressed time base in the serial communication simulation and manually correct the timing later. For this particular project, serial communication simulation did not appear to be a subject to pursue.

IV. EXPERIMENTAL RESULTS

The first proof required of any computer program is whether or not it indeed does perform its intended functions. Demonstrating this fact for the preprocessor developed for this thesis actually involves two factors. The preprocessor must perform the functions of an assembler and initiate the simulation. While doing this, it must also demonstrate the fidelity for which it was optimized.

The initial pages of output from a run of the program are presented in Figures 6, 7, and 8. The symbol table and the assembled listing of the program to be simulated are presented in Figure 6. The loadable version of the program, with associated location counter values, is shown below the listing. This simple program utilizes an interrupt driven routine, triggered by the 8255 chip, to retrieve and store an externally-input character. The main routine uses the control word function and the output function of the 8255, as well as providing a main processing stream to be interrupted.

Figures 7 and 8 are the initial sections of the output from the simulation, triggered by the preprocessor after it has loaded the assembled program into the simulated memory. The individual entries give the hexadecimal values of selected registers during each clock cycle. The changes in these registers, established in the hardware definition section of the simulator program, trace the execution of the program. The output presented illustrates the type of information collected.

```
;*****************************************************************
;**                                                             **
;**      END OF TRANSLATION, BEGIN SIMULATION                   **
;**                                                             **
;*****************************************************************
$SIMULATE
*OUTPUT   LABEL(1,2)=Y,X,A,DATABUF,ADDBUFFER,INEN,INT,
          IR,PC,TEMP,STAT,CWORD,CPORT,APORT,INT1,INT2,INTR
*SWITCH   1,INTST=ON
*SWITCH   2,INIT=ON
*SWITCH   15,INDATA=ON
*LOAD

ASM 8080    MEM
LIST
NORG
                    ASSEMBLY BEGINS HERE

          SYMBOL TABLE

     SYMBOL INPUT     VALUE 0038
     SYMBOL SETUP     VALUE 0100
     SYMBOL OUTPUT    VALUE 0105

                    PASS TWO

LC     CODE                          PROGRAM STATEMENTS

0000   00  C1                 DW      0C100H          ;STORED DATA
                            ORG     56
                    ;       INTERRUPT HANDLER ROUTINE
0038   3E  06       INPUT:  MVI     A,06            ;KILL INTR SIGNAL
003A   D3  03               OUT     3
003C   3E  08               MVI     A,08H           ;KILL I8FA
003E   D3  03               OUT     3
0040   DB  00               IN      0               ;RETRIEVE INPUT CHARACTE
0042   32  01  02           STA     0201H           ;STORE IT
0045   FB                   EI                      ;REENABLE INTERRUPT
0046   C9                   RET                     ;RETURN
                            ORG     0100H
                    ;               MAIN ROUTINE
0100   FB           SETUP:  EI                      ;ENABLE INTERRUPT
0101   3E  09               MVI     A,09            ;SET INT2
0103   D3  03               OUT     3
0105   3A  00  02   OUTPUT: LDA     0200H           ;LOAD CHARACTER
0108   D3  00               OUT     0               ;OUTPUT TO PORT A
010A   76                   HLT
                            ORG     0200H
0200   C2  00               DW      00C2H           ;STORED INFO
                            END

MEM (:0000-]=:00,:C1
MEM (:0038-]=:3E,:06,:D3,:03,:3E,:08,:D3,:03,:DB,:00,:32,:01,:02,:FB
MEM (:0046-]=:C9
MEM (:0100-]=:FB,:3E,:09,:D3,:03,:3A,:00,:02,:D3,:00,:76
MEM (:0200-]=:C2,:00

     ***** END OF ASSEMBLER ROUTINE   *****

     ASM 8080    MEM
     PC=:0100
    *SIN         800,3
    END OF DATA ON INPUT
```

Figure 6. Test Program Listing

OUTPUT OF SIMULATION

```
**** SWITCH INTERRUPT ****
    INTS = ON
     Y = ...6      X = ...6      A = ..00   DATA = ..00   ADDB = 0000   INEN = ...0
    IR = ..00     PC = 0100   TEMP = ..00   STAT = ..00   CWOR = ..00   CPOR = ..00
  INT1 = ...0   INT2 = ...0   INTR = ...0
*************************************************************************************
LABEL CYCLE    1              TRUE LABELS                    CLOCK CYCLE    1
                             /READ*P(0)/
                             /ACKA'*OBFA'
                             /CSLO'*P(0)/
*************************************************************************************
**** SWITCH INTERRUPT ****
    INIT = ON
     Y = ...1      X = ...1      A = ..00   DATA = ..00   ADDB = 0000   INEN = ...0
    IR = ..00     PC = 0100   TEMP = ..00   STAT = ..00   CWOR = ..00   CPOR = ..54
  INT1 = ...0   INT2 = ...0   INTR = ...0
*************************************************************************************
LABEL CYCLE    2              TRUE LABELS                    CLOCK CYCLE    1
                             /M(1)*T(1)*P
     Y = ...1      X = ...1      A = ..00   DATA = ..00   ADDB = 0100   INEN = ...0
    IR = ..00     PC = 0100   TEMP = ..00   STAT = ..00   CWOR = ..00   CPOR = ..54
  INT1 = ...0   INT2 = ...0   INTR = ...0
*************************************************************************************
LABEL CYCLE    3              TRUE LABELS                    CLOCK CYCLE    2
                             /SYNC*P(0)/
*************************************************************************************
LABEL CYCLE    4              TRUE LABELS                    CLOCK CYCLE    2
                             /STST*P(1)/
     Y = ...1      X = ...1      A = ..00   DATA = ..A2   ADDB = 0100   INEN = ...0
    IR = ..00     PC = 0100   TEMP = ..00   STAT = ..A2   CWOR = ..00   CPOR = ..54
  INT1 = ...0   INT2 = ...0   INTR = ...0
*************************************************************************************
LABEL CYCLE    5              TRUE LABELS                    CLOCK CYCLE    3
                             /DBIN*NMR*P
*************************************************************************************
LABEL CYCLE    6              TRUE LABELS                    CLOCK CYCLE    3
                             /DBIN*NMR*P
     Y = ...1      X = ...1      A = ..00   DATA = ..FB   ADDB = 0100   INEN = ...0
    IR = ..00     PC = 0100   TEMP = ..00   STAT = ..A2   CWOR = ..00   CPOR = ..54
  INT1 = ...0   INT2 = ...0   INTR = ...0
*************************************************************************************
LABEL CYCLE    7              TRUE LABELS                    CLOCK CYCLE    4
                             /READ*P(0)/
*************************************************************************************
LABEL CYCLE    8              TRUE LABELS                    CLOCK CYCLE    4
                             /M(1)*T(2)*P
     Y = ...1      X = ...2      A = ..00   DATA = ..FB   ADDB = 0100   INEN = ...0
    IR = ..00     PC = 0101   TEMP = ..00   STAT = ..A2   CWOR = ..00   CPOR = ..54
  INT1 = ...0   INT2 = ...0   INTR = ...0
*************************************************************************************
LABEL CYCLE    9              TRUE LABELS                    CLOCK CYCLE    5
                             /READ*P(0)/
*************************************************************************************
LABEL CYCLE   10              TRUE LABELS                    CLOCK CYCLE    5
                             /M(1)*T(3)*P
     Y = ...1      X = ...3      A = ..00   DATA = ..FB   ADDB = 0100   INEN = ...0
```

Figure 7. Simulator Output, Page 1

```
   IR = ..FB      PC = 0101   TEMP = ..00   STAT = ..A2   CMDR = ..00   CPDR = ..54
 INT1 = ...0   INT2 = ...0   INTR = ...0
*****************************************************************************************
LABEL CYCLE - 11              TRUE LABELS                   CLOCK CYCLE    6
                             /READ*P(0)/
*****************************************************************************************
LABEL CYCLE - 12              TRUE LABELS                   CLOCK CYCLE    6
                             /M(1)*T(4)*P
    Y = ...1      X = ...0      A = ..00   DATA = ..FB   ADDR = 0100   INEN = ...1
   IR = ..FE      PC = 0101   TEMP = ..00   STAT = ..A2   CMDR = ..00   CPDR = ..54
 INT1 = ...0   INT2 = ...0   INTR = ...0
*****************************************************************************************
LABEL CYCLE - 13              TRUE LABELS                   CLOCK CYCLE    7
                             /READ*P(0)/
*****************************************************************************************
LABEL CYCLE - 14              TRUE LABELS                   CLOCK CYCLE    7
                             /M(1)*T(1)*P
    Y = ...1      X = ...1      A = ..00   DATA = ..FB   ADDR = 0101   INEN = ...1
   IR = ..FE      PC = 0101   TEMP = ..00   STAT = ..A2   CMDR = ..00   CPDR = ..54
 INT1 = ...0   INT2 = ...0   INTR = ...0
*****************************************************************************************
LABEL CYCLE - 15              TRUE LABELS                   CLOCK CYCLE    8
                             /SYNC*P(0)/
*****************************************************************************************
LABEL CYCLE - 16              TRUE LABELS                   CLOCK CYCLE    8
                             /STST*P(1)/
    Y = ...1      X = ...1      A = ..00   DATA = ..A2   ADDR = 0101   INEN = ...1
   IR = ..FE      PC = 0101   TEMP = ..00   STAT = ..A2   CMDR = ..00   CPDR = ..54
 INT1 = ...0   INT2 = ...0   INTR = ...0
*****************************************************************************************
LABEL CYCLE - 17              TRUE LABELS                   CLOCK CYCLE    9
                             /OBIN*NMR'*P
*****************************************************************************************
LABEL CYCLE - 18              TRUE LABELS                   CLOCK CYCLE    9
                             /OBIN*NMR'*P
    Y = ...1      X = ...1      A = ..00   DATA = ..3E   ADDR = 0101   INEN = ...1
   IR = ..FE      PC = 0101   TEMP = ..00   STAT = ..A2   CMDR = ..00   CPDR = ..54
 INT1 = ...0   INT2 = ...0   INTR = ...0
*****************************************************************************************
LABEL CYCLE - 19              TRUE LABELS                   CLOCK CYCLE   10
                             /READ*P(0)/
*****************************************************************************************
LABEL CYCLE - 20              TRUE LABELS.                  CLOCK CYCLE   10
                             /M(1)*T(2)*P
    Y = ...1      X = ...2      A = ..00   DATA = ..3E   ADDR = 0101   INEN = ...1
   IR = ..FE      PC = 0102   TEMP = ..00   STAT = ..A2   CMDR = ..00   CPDR = ..54
 INT1 = ...0   INT2 = ...0   INTR = ...0
*****************************************************************************************
LABEL CYCLE - 21              TRUE LABELS                   CLOCK CYCLE   11
                             /READ*P(0)/
*****************************************************************************************
LABEL CYCLE - 22              TRUE LABELS                   CLOCK CYCLE   11
                             /M(1)*T(3)*P
    Y = ...1      X = ...3      A = ..00   DATA = ..3E   ADDR = 0101   INEN = ...1
   IR = ..3E      PC = 0102   TEMP = ..00   STAT = ..A2   CMDR = ..00   CPDR = ..54
 INT1 = ...0   INT2 = ...0   INTR = ...0
*****************************************************************************************
LABEL CYCLE - 23              TRUE LABELS                   CLOCK CYCLE   12
                             /READ*P(0)/
*****************************************************************************************
```

Figure 8. Simulator Output, Page 2

Due to the extremely large amount of data that is produced, only these samples are shown.

As stated earlier, a measure of the fidelity of a simulation can be made using a timing analysis. Inspection of the microoperation sequences can show that the individual operations correspond to the target machine, but only a timing analysis can demonstrate the integration of the system as a whole. A timing analysis also serves to highlight any timing irregularities inserted by the mechanics of the simulation. An example of this type of analysis, using the simulation developed for this project, is presented in Figure 9. The figure lists the assembly language program run by the simulation and presents an accounting and comparison of the timing factors.

The analysis illustrates two of the irregularities of the simulation that were discussed earlier in the paper. The first is the extra clock cycle added to all immediate operations, such as MVI (MoVe Immediate). The alternative to this added cycle was to create a separate routine for each immediate operation, an alternative judged to be far less acceptable. The second factor shown is the presence of two clock cycles added to each memory and I/O access. As explained earlier, this factor is introduced by the software timing mechanism. Even though the mechanism is not updating the machine cycle and state numbers during an access, the master clock must continue to run to provide execution timing. The cycle and state numbers remain correct, but the clock cycle timing must include a correction factor to account for the extra cycles. The analysis must also account for program dependent conditions. The interrupt generated in the execution of this program

Operation	Cycle Time	Factor	Total
PUSH PSW	10	6	16
MVI	7	4+1	12
OUT	10	6	16
MVI	7	4+1	16
OUT	10	6	16
IN	10	6	16
STA	13	8	21
POP PSW	10	6	16
EI	4	2	6
RET	10	6	16
EI	4	2	6
MVI	7	4+1	12
OUT	10	6	16
LDA	13	8	21
OUT	10	6	16
HLT	7	2	9
			224

Operation times	224
Interrupt time	14
Startup time	1
	239 *clock cycles*

Figure 9. Timing Analysis

is handled by the 8228 module as a RESTART 7 instruction, producing 14 clock cycles that have no apparent source in the program code. As shown in the figure, all of these times may be added together to produce a time estimate, measured in clock cycles. This estimate agrees exactly with the timing of the simulation run of the program.

V. CONCLUSIONS

The project described in this paper is mainly the proof of a concept. The ability of the CDL simulator to accept the integration of a preprocessor and faithfully simulate a microprocessor-based system is evaluated by attempting an implementation of those tasks. The effort was directed at making the implementation succeed rather than making it highly practical. Yet the practicality of this simulation is certainly one of its strongest assets.

Certainly, the first candidate for application of this package is hardware simulation, the most common use of simulator packages. Simulation permits the comparison of alternate constructs at any level, from single devices to system architectures, to provide performance data without the investment and time penalty of actual hardware construction. Such a process can be used to fine tune a system for a specific application. The simulation of software is a less obvious candidate for application, but the same refinement process can be used to view program execution on a time-phased, register-transfer level. Such refined software would be useful for the highly compact, intensely iterative programs normally stored in read-only memory for process control or communications handling devices.

To facilitate application of this simulator, there are several improvements that can be made. These improvements range in difficulty from major revisions to relatively simple extensions of the existing

program. Language versatility is one of the simple extensions. The preprocessor, as currently written, will service only the Intel 8080 assembly language. However, the necessary mechanisms for choice of a language set are already included in the preprocessor program. The alternative language would have to be reduced to a table format compatible with the preprocessor. Pseudo-operation routines would have to be written and included in the already stored subroutine. Finally, alterations would have to be made to the routines for operand interpretation if the conventions of the desired alternative differed substantially from those of the 8080 assembly language. Due to the increased storage requirements for these alternative user-selected options, the most effective implementation of these features might be to compile complete versions of the simulator package for each language to be used and have them user-selected as a part of basic program selection. This method would allow versatility without sacrificing program compactness.

Another avenue for improving the simulation lies in that of simulator expansion. The current version of this program proves that system simulation is feasible. To make the simulator more useful, a library of module and device simulation routines could be developed. The hardware modules and microprocessor simulations could be selected to produce the desired system configuration. Addition of an assembly language program for the target processor would complete the system simulation, ready for input in the simulator.

The greatest return in efficiency could be reaped after the greatest effort in program improvement: restructure. The current program

is time consuming not only because it is so intensely iterative, but because it suffers from the time penalties imposed by its base language and structure. FORTRAN shares the algebraic format of CDL, but the deeply nested subroutine calls and complex logic used in the simulator do not lend themselves to time-efficient computation. The use of structured programming could help to streamline the sequence of subprograms being called and avoid some of the machine overhead involved in those calls, even at the expense of some redundant coding in different routines. The use of a structured language, such as PASCAL, could produce even more comprehensive changes. Constructs such as the CASE statement could replace sections of decoding logic and drastically reduce execution time while improving program flow. The bit manipulation functions lacking in FORTRAN could possibly be incorporated through the alternate language, eliminating the necessity for sizable assembly language subroutines to perform those functions. The resulting program unification would certainly be a significant achievement. A restructured program might also be able to handle concurrent processes with greater ease by eliminating the need to evaluate every conditional microstatement on every iteration of the program. As stated before, the effort involved in a restructure is extensive, but the resulting improvements in utility and computational speed would be most impressive.

Simulation is an important tool in system design. Its merit rests in its ability to save money and effort by providing results of tests on system configurations that exist only on paper. The simulation package developed in this project attempts to combine the important

user-oriented features, high level of detail, and easily interpretable simulation results. There are ample opportunities to use the system as it exists and system improvements options exist at various levels of effort. The possibilities of microprocessor-based system simulation are limited only by the imagination and energy of the user.

REFERENCES

1. Chu, Yaohan, "Introducing the Computer Design Language," Digest of Papers, Comcon 72, San Francisco, September, 1972, pp. 215-218.

2. Chu, Yaohan, Computer Organization and Microprogramming, Prentice-Hall, 1972.

3. Cwik, Terry T., Multiprocessing Simulation of the Intel 8080 and the PDP-8 Using Computer Design Language, Auburn University, Auburn, Alabama, 1976.

4. Donovan, John J., Systems Programming, McGraw-Hill, New York, 1972.

5. MCS-80 User's Manual, Intel Corporation, Santa Clara, CA, October 1977, pp. 2-16 to 2-19.

6. Ibid., pp. 6-1 to 6-38.

7. Ibid., pp. 6-223 to 6-243.

APPENDIX A

CDL SIMULATOR

USER'S MANUAL

FOREWARD

This manual is based mainly upon information presented in the original user's manual compiled by Terry Cwik. The manual was rewritten and restructured to include material on the functional description of the CDL simulator as well as its syntax and to improve the clarity of the original manual. The user's manual for the CDL simulator preprocessor was also added.

Syntax in this manual is presented in a standard notation. Formats are presented on a line separate from the text. Upper case items refer to entries which must be made exactly as shown. Lower case items refer to types of entries only. All delimiters, such as slashes and parentheses, are considered significant and required.

TABLE OF CONTENTS

Introduction

Computer Design Language (CDL) was originally designed by Dr. Yaohan Chu in 1965. It was designed to represent the architecture and operation of computer hardware at the register transfer level, using an algebraic notation. The language is versatile enough to serve two major purposes. CDL can serve as a standard language for defining the structure of digital systems, especially in an instructional setting. The language, used with a simulator program, can also be used in the simulation of existing digital systems or in the testing and development of new systems. This handbook is intended as an aid in using CDL in this second manner, with an incorporated simulator program.

CDL Structure

The CDL simulator program works in several logical steps. The first step is accomplished by the translator section. The logical design of the subject hardware, written in CDL, is read into the host computer as card images. The translator converts the hardware design, in the form of declaration statements, into a form suitable for computer manipulation, namely groups of tables and a pseudo program called the Polish string.

This information is passed to the simulator section, composed of five routines. The loader routine accepts programs and data to be loaded into the simulated memory or specified registers in the design. The simulator routine controls the execution of the test program. The switch routine incorporates the options of manual switch settings. The output routine controls the identity and frequency of output values

produced by the simulation. The simulation may be reinitialized for another test by the reset routine.

Translator Section

The first task in using CDL for simulation is to specify the design of the selected logical circuit in CDL terms. This specification normally occurs in two phases. In the definition phase, the hardware architecture of the system is stated. In the operational phase, the logical actions of the system are defined at the register transfer level. The definition phase consists mainly of declaration statements, defining the hardware elements as variables, so that they can be used in expressing the operation phase statements.

Declaration Statements. These statements are used to define basic hardware units. The following devices are defined in CDL:

REGISTER	SWITCH
SUBREGISTER	TERMINAL
MEMORY	BLOCK
DECODER	CLOCK
LIGHT	BUS

The first four characters of each device name are significant to the simulator. The syntax of the declaration statement is

device name, list

The device name begins in column two and the comma trailing the device name is required. The devices are discussed in more detail below.

REGISTER Declaration. An individual register is defined by a name and a number in parentheses. This number defines the length and order of the bit positions. Default value of the number is a single bit. Examples are presented in Figure A-2.

SUBREGISTER Declaration. This declaration identifies a section of a previously declared register. The declared register, followed by the subregister name, is equated to a certain string of bits within that register. Subregister names must be unique to the four significant characters, even when referenced to different registers. Examples are presented in Figure A-1.

MEMORY Declaration. A memory is referenced by its name and a previously declared register which will be its address register. The range of the address and the bit order of the words in the memory are specified. Thus,

MEMORY, M(R) = M(0-99, 7-0)

defines a 100 byte memory space named M.

DECODER Declaration. This declaration defines a device which equates each value of the contents of all or a section of a previously defined register to a single output. The decoder's name and range of values is equated to the register or section of a register. Examples are presented in Figure A-1.

CLOCK Declaration. A clock is defined for the purpose of event synchronization. It can only be referenced in a label expression, to be defined later. The clock is defined by its name and a number, one

less than the number of discrete timing levels desired. Examples are presented in Figure A-1.

SWITCH Declaration. An external switch condition can be simulated by this declaration. It is defined by the switch name and possible positions, initial position first. A maximum of ten switch positions is permitted. An example of the definition format would be

SWITCH, STRT (OFF, ON), TEMP (T1, T2, T3).

In use a switch may be either set or read. To set a switch, the name is equated to the desired position, such as STRT = ON. A switch is read, giving a value of 1 or 0, by citing the switch and a position, such as STRT (ON).

TERMINAL Declaration. Logical networks or multiple references for a single device are handled by the TERMINAL declaration. The terminal is simply defined in terms of previously declared devices. Its use may be very similar to a DECODER declaration. Examples are presented in Figure A-1.

LIGHT Declaration. Panel lights may be included by using this declaration. As in the SWITCH declaration, the light is named and its states given, initial state first.

LIGHT, RUN (OFF, ON), PWR (ON, OFF)

is a typical example. The set and read options also follow the form of the SWITCH.

BUS Declaration. A bus is defined in terms of its width in lines, as in BUS, DATA (0-7), ADDR (0-15).

BLOCK Declaration. This construct is actually a software mechanism, similar to a subroutine. The BLOCK name serves as a title for a group of microstatements, as defined below. The microstatements are enclosed in parentheses, with nesting and such options as IF, THEN, ELSE allowed. This group of statements is called to be executed by a DO statement, in the form

DO/block name.

Thus

BLOCK, SWAP (A=B, B=A)

would be called by

DO/SWAP.

Micro Statements

Once the hardware architecture has been defined, the logic functions impressed on these elements are defined using microstatements. The basic form of a microstatement is

variable = expression

An expression is a group of variables and their associated operators.

The standard operators listed in Table A-1 are available for use in microstatements. Special operators may be defined by the user in a separate subprogram. This subprogram is of the form

```
*OPERATOR, first argument,name,second argument
// operations comprising the function of the
operator, RETURN END
```

Argument names must include bit structure if over one bit. The second argument is necessary only for binary operators. The blank label, //, will cause immediate execution of the listed operations when

the operator is invoked by its name. The subprogram is terminated by the RETURN and END. Table A-1 also lists several special operators built into the simulation program.

Microstatements have several forms. An unconditional microstatement is of the form

variable = expression.

The effect of this construct is to replace the named variable, a storage element, with the result of the expression. The named variable, either a device or a part of a device, must not be replaced more than once in any set of microstatements to be performed during a single cycle.

A conditional microstatement is of the form

IF (expression) THEN (microstatements).

If the expression contained in the parentheses following the IF is true, thus equal to 1, then the microstatements following the THEN are executed; otherwise, they are simply skipped. This form may be extended to the form

IF (expression) THEN (microstatements) ELSE (microstatements).

Execution is identical to the first form, except that when the expression is false, the microstatements following the ELSE are executed. These forms may be nested by using the precedence rules of parentheses. This nesting capability can be used to design complex and powerful decision functions.

Microstatements are used to build other types of statements. The switch statement has the form

/ switch name (position) / microstatements.

If the named switch is in the indicated position, the microstatements are executed; otherwise, they are not. This construct simulates the sensing of switch positions.

The most common statement in simulations is the label statement. It has the form

/ label / microstatements

where a label is the logical AND of an expression and a clock level. The expression must not include a reference to a clock level. When the expression and the clock level are both logically true, the microstatements are executed. This construct simulates the execution of time-phased logic.

Finally there is the end statement. The word END indicates the physical end of the statements defining a system design. It terminates the translation process and causes control to pass to the simulator routines.

Simulator Section

Once the hardware and operational definitions have been made, the simulator is prepared to execute the test program. The execution is carried out in a loop of processes called the label cycle. During each cycle, four tasks are performed. First, if any switch action is designated to occur in the current label cycle, the executable statements that it activates will be performed. Secondly, all label values are evaluated and those with true label expressions are noted. Third, the statements corresponding to the true labels are executed. All values resulting from these statements are evaluated, collected, and

then stored. Fourth, it is determined if the simulation should be terminated at this point. If not, the next label cycle is begun. If it is terminated, a RESET routine may be called to begin another simulation.

Syntax

As with all computer programs, there are syntax rules which must be obeyed if the program is to function as specified. There are general syntax rules for the use in all statements and control cards to direct the sequencing of the simulator program; the Job Control cards necessary to run this program will be considered separately.

General Syntax

Variables. A variable must be defined in a declaration statement before it can be used elsewhere. A variable may consist of one to four characters. The first character must be alphabetic. Embedded blanks and special characters other than "+", "-", "," "*;, :/", ".", "'", "$", or "=" are simply ignored and dropped. Longer variable names may be used, but the translator uses only the first four significant characters. Thus "START1", "START2", and "STAR" are all treated as "STAR" by the simulator. The following words are reserved and must not be used as variable names: IF, THEN, ELSE, DO, CALL, RETURN, and END.

Constants. Three forms of numerical constants are available for use. A hexadecimal constant, denoted by a colon preceding its digits, is accepted up to a maximum of eight digits. A binary constant, denoted by a semicolon preceding its digits, is accepted up to a maximum

of 32 digits. A decimal constant, denoted by no delimiter, is accepted up to a maximum of nine digits. Blanks, special characters other than those listed above, and characters outside the set permissible for the particular form are ignored and dropped.

Continuations. Declaration statements are continued to subsequent cards by placing a "1" in column one of the subsequent cards. Label and Switch statements are continued to subsequent cards by leaving column one blank. All statements are limited to 250 terms, where a term is considered to be either a variable, a constant, or a valid special character.

Comment Cards. Placing a "C" in column one will produce a comment line, ignored by the translator. Placing a "C" in column one of subsequent cards allows continuation of the comment.

Card Format. Declaration statements, labeled statements, and END statements may be punched anywhere in columns two through 72. Column one is used only for comments and continuations. Free use of blanks is permitted and is encouraged to promote readability.

Control Cards. Control cards are used to call the functional elements of the simulation system into action. These cards will be discussed in the order in which they will normally be encountered.

Translator. The translator is called first to translate the design information into a form suitable for simulation by the program. The first column contains the control symbol "$", followed by the control word TRANSLATE or TRANS. The translator will retain control until the next card with the control symbol in column one is read. The design deck must begin with the control card (MAIN, where the se-

condary control symbol "*" appears in column one. The design deck is terminated using an End card, with END in columns one through three. If special operators are to be defined, they are separated from the rest of the translation. The special operator defintions are all started with the *OPERATOR card and closed with the END card.

Simulator. Control is next passed to the simulator by the $SIMULATE card, with the control symbol in the first column. Asterisk control cards are used to pass control between the simulator's five routines: Output, Switch, Load, Simulate, and Reset. Unlike the preceding example, END cards are not necessary to separate sections.

The Output routine specifies the format of the printed output of the simulation. The format of the control card is as follows:

columns 1-7	*OUTPUT
columns 11-15	CLOCK or LABEL
columns 16-21	(n,m)=
columns 22-72	list

The CLOCK or LABEL designation controls whether data is output on clock cycles or label cycles, beginning on the nth cycle and repeating every mth cycle thereafter. The list following specifies the registers, memory locations, and other devices whose value is to be output each time. Continuation cards for the list are permissible as long as column one is left blank. All output values are listed in hexadecimal format, regardless of input format.

The Switch routine allows the simulation of manual switch settings. A separate card is necessary for each switch action. It has the following format:

columns 1-7	*SWITCH
columns 11-12	n,
column 13	switch name = switch position

The number n specifies the label cycle before which the switch action occurs. The switch name and its position must have been declared previously. In the output, each switch action will cause an output with a heading which states that the switch action has occurred.

The Load routine stores test programs and data in memory and registers. The *LOAD card precedes the data cards. Data cards use columns 2 through 72, with free use of blanks permitted. There are no continuation cards. Each card must be begun in column 2 and be self-sufficient. A data card may contain a number of lists, separated by commas. Only declared full registers and full memory locations may be loaded. The format for the two types of entries are different. Registers are loaded with the format

"register name = n",

where n is the value to be loaded. There are three variations of the format for loading memory locations. Single memory locations can be loaded in the form

M(m) = n,

where M(m) denotes location m of memory M and n denotes the value to be loaded. Multiple consecutive locations can be loaded in the form

M(m1-mx) = n1,n2,...,nx,

where locations 1 through x are loaded with values n1 through nx. The ending address may also be implied rather than stated in the form

M(m1-) = n1, n2,...,ny,

where consecutive memory locations are loaded, beginning with m1 and continuing until y locations are filled. There is a software imposed limit of 80 load entries.

The Simulate routine initiates the actual simulation subprogram. The control card specifies the simulation termination parameters. It has the following format:

columns 1-4	*SIM
columns 11-	n,m

The number n specifies the maximum number of label cycles to be generated. The number m specifies the maximum number of consecutive label cycles to be allowed without a change in the active labels. When m label cycles have passed with no changes, the simulation is automatically terminated.

The Reset routine performs reinitialization of the simulator subprogram to allow another run of the simulator on the same design. The control card has the following format:

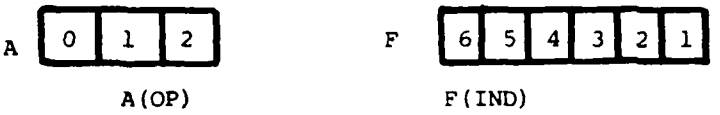
columns 1-6	*RESET
columns 11-	options

The options are one or more of the following terms, separated by commas. CLOCK resets the clock cycle only. CYCLE resets the label cycle counter and the clock cycle counter. OUTPUT resets the previously requested output parameters, just as SWITCH resets the previously requested manual switch operations. In both cases, another *OUTPUT or *SWITCH card is expected. The next simulation will begin with another *SIM card.

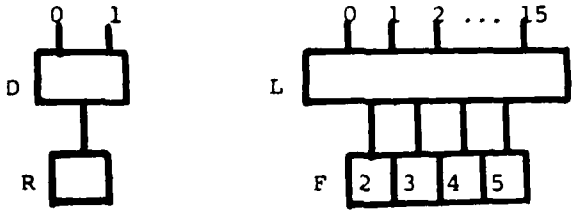
A typical simulation with all internal control cards appears in Figure A-2, depicting a single simulation run. While these internal cards are uniform, external control cards are unique for each site. The job control cards necessary to use the CDL program stored in a given system must be obtained.

REGI, A(0,2), R, F(6-1)

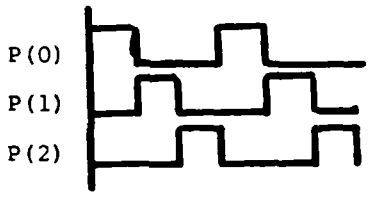
SUBR, A(OP)=A(1,2), F(IND)=F(6-4)



DECO, D(0-1)=R, L(0-15)=F(2-5)



CLOCK, P(2)



TERMINAL, C(0-2)=A(0-2)', D=(B(0)+B(1))

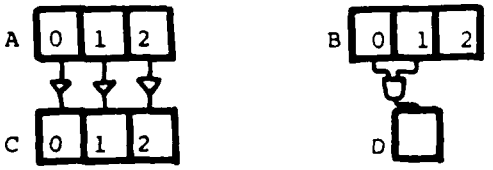


Figure A-I. CDL Device Examples

```
C ***********************************************************
C **                                                       **
C **      END OF TRANSLATION, BEGIN SIMULATION             **
C **                                                       **
C ***********************************************************
$SIMULATE
*OUTPUT   LABEL(1,2)=Y,X,A,DATABUF,ADDBUFFER,INEN,INT,
          IR,PC,TEMP,STAT,CWORD,CPORT,APORT,INT1,INT2,INTR
*SWITCH    1,INTST=ON
*SWITCH    2,INIT=ON
*SWITCH    75,INDATA=ON
*LOAD
```

```
ASM 8080    MEM                                                  00005550
LIST                                                             00005560
NORG                                                             00005570
```

ASSEMBLY BEGINS HERE

SYMBOL TABLE

```
SYMBOL INPUT    VALUE 0038
SYMBOL SETUP    VALUE 0100
SYMBOL OUTPUT   VALUE 0105
```

PASS TWO

```
LC     CODE                          PROGRAM STATEMENTS
0000   00  C1                        DW      0C100H          ;STORED DATA
                                     ORG     56
                                *    INTERRUPT HANDLER ROUTINE
0038   3E  06            INPUT:     MVI     A,06            ;KILL INTR SIGNAL
003A   D3  03                       OUT     3
003C   3E  08                       MVI     A,08H           ;KILL IBFA
003E   D3  03                       OUT     3
0040   DB  00                       IN      0               ;RETRIEVE INPUT CHARACTE
0042   32  01  02                   STA     0201H           ;STORE IT
0045   FB                           EI                      ;REENABLE INTERRUPT
0046   C9                           RET                     ;RETURN
                                     ORG     0100H
                                *    MAIN ROUTINE
0100   FB            SETUP:     EI                      ;ENABLE INTERRUPT
0101   3E  09                   MVI     A,09            ;SET INT2
0103   D3  03                   OUT     3
0105   3A  00  02    OUTPUT:    LDA     0200H           ;LOAD CHARACTER
0108   D3  00                   OUT     0               ;OUTPUT TO PORT A
010A   76                       HLT
                                     ORG     0200H
0200   C2  00                       DW      00C2H           ;STORED INFO
                                     END
```

```
MEM (:0000-)=:00,:C1
MEM (:0038-)=:3E,:06,:D3,:03,:3E,:08,:D3,:03,:DB,:00,:32,:01,:02,:FB
MEM (:0046-)=:C9
MEM (:0100-)=:FB,:3E,:09,:D3,:03,:3A,:00,:02,:D3,:00,:76
MEM (:0200-)=:C2+00
```

```
*****  END OF ASSEMBLER ROUTINE   *****

ASM 8080    MEM
PC=:0100
*SIM       800,3
END OF DATA ON INPUT
```

Figure A-2. Simulation with Control Cards

STANDARD OPERATORS

SYMBOL	FUNCTION	EXAMPLE	EXPLANATION
' (apostrophe)	Complement	A'	Logical NOT a
= (equal sign)	Replace	A=B	Contents of A are replaced by contents of B
- (dash)	Concatenate	A-B	Contents of A and B are placed side by side
+ (plus sign)	Logical OR	A+B	Bit by bit OR, where A and B must be conformal
* (asterisk)	Logical AND	A*B	Bit by bit AND, where A and B must be conformal
.EQ.	Equality function	A.EQ.B	Gives '1' if A and B are equal, '0' if not
.NE.	Inequality function	A.NE.B	Gives '1' if A and B are unequal, '0' if they are not

SPECIAL OPERATORS

.ERA.	Exclusive OR	A.ERA.B	Exclusive OR of A and B
.ADD.	Sum	A.ADD.B	Algebraic sum of A and B, with overflow bit discarded
.SUB.	Difference	A.SUB.B	Algebraic sum of A and NOT B, with overflow bit discarded
.COUNT.	Increment	A.COUNT.	Adds 1 to A, with overflow bit discarded
.LT. .LE. .GE. .GT.	Magnitude operators	A.LT.B A.LE.B A.GE.B A.GT.B	Gives '1' if algebraic conditions (less than, less or equal, greater or equal, greater than) are met, '0' if they are not met

Table A-1. Standard and Special Operators

APPENDIX A-1

CDL SIMULATOR PREPROCESSOR

USER'S MANUAL

This manual is designed to present the rules and constructs governing the operation of the preprocessor option of the CDL simulator program. Corresponding information on the features of the simulator itself is contained in the basic user's manual. Familiarity with the simulator is assumed for the reader of this manual.

TABLE OF CONTENTS

PREPROCESSOR USER'S MANUAL

Format

The standard unit of input to the preprocessor is the card image. The preprocessor is written in FORTRAN, a language with limited bit manipulation capabilities. The card image, therefore, must be highly structured. It consists of a number of fields, some of which are optional. Violation of the format will result in a printed error message and a termination of the processing at the next logical break point.

Assembler Options

The first information given to the preprocessor must be the user's choice of the available options. The first card of the load module contains three fields. The first field, columns 2 through 5, must contain the directive "ASM" in order to invoke the assembler. The second field, columns 6 through 9, contains the assembly language option. The original version of the preprocessor responds only to the choice "8080" which invokes the Intel 8080 assembly language. The growth option to alternative languages is provided in the preprocessor code. The third field, columns 14 through 17, is reserved for the name of the memory device where the object code is to be stored. This device name must have been declared earlier in the hardware definition portion of the simulation program. The name may contain more than four letters, but as in the CDL simulator, only the first four are significant.

The second card contains a single field, columns 2 through 5, for the symbol table listing option. The directive "LIST" will cause the symbol table to be printed out at the beginning of the assembler output. The directive "NOLIST" will suppress the printing of that table. As with the memory name declaration, only the first four letters of that directive are significant. This card and the one following it are designed as single cards to provide for easy changes of those options that are likely to be changed.

The third card is the location counter initialization card. The single field, columns 2 through 5, may contain one of two directives. "NORG" specifies a location counter of zero. "ORG=" followed by the expression beginning in column 6 will set the location to the value of that expression. The expression is delimited by the first blank encountered. This card is followed by the assembly language program.

Assembly Language Program

The standard line of the assembly language program is of the form:

```
label:  opcode              operand, operand  ; comment
```

The first field of the card image is the label field. This optional field begins in column 2. It may contain a maximum of six alphanumeric characters, the first of which must be alphabetic. The field terminates with a colon.

The second field is the opcode field. This required field begins in column 10. It contains a maximum of four alphabetic characters, terminating in a blank. The contents of this field must match the

mnemonic opcodes stored in the assembly language table being used by the assembler or an error will be generated.

The third field is the operand field. This field may be required based on the requirements of the preceding opcode. The field may contain alphanumeric characters, labels, or expressions terminated by a blank. There can be no embedded blanks. If two operands are required, they are separated by a comma.

The fourth field is the comment field. It begins immediately after the blank terminating the operand field. If there is no operand present, it begins in column 22. While no delimiter is required to separate it from the preceding text, a semicolon is suggested to improve readability. The line must end by column 72.

An entire line of comment can be entered in the place of a line of program code by inserting an asterisk in column 1. The following line receives no processing.

Data Types

The assembler supports six basic data types. The format of each is specified as follows.

Decimal data. Each decimal number contains only numerics. Examples: 14, 17.

Hexadecimal data. Each hexadecimal number must begin with a numeric digit and must be followed by the letter H. Examples: 9B7H, 0AFH.

Binary data. Each binary number must be followed by the letter B. Examples: 1101B, 011B.

Octal data. Each octal number must be followed by the letter O. Examples: 720, 550.

Character data. Character data may be introduced, mainly via the DB or DW pseudo-ops. Data strings must be delimited at both ends by single quotes. Further coding rules are included in the discussion of the DB and DW directives. Examples: 'HELLO', 'CHAPTER 2'.

Expressions. Only simple expressions involving addition and subtraction are supported by the assembler. No leading minus signs or embedded blanks are permitted. No special delimiters are used for expressions, which are terminated by the first blank encountered. Label data may be used in addition to numeric values. Numeric data alone is considered as a simple expression. Examples: LABEL+3, 14, SUM-2+TAX.

Pseudo-ops

The assembler, in addition to machine operation codes, supports certain pseudo-op codes or assembler directives which control the assembler as it generates object code. The mnemonics for these directive commands are included in the operation code table of the preprocessor, with a flag that identifies them as pseudo-ops and transfers control to a routine performing the necessary functions. These functions are of several types.

Data definition. The DB (Define Byte) and DW (Define Word) directives define data to be entered into storage locations. DB stores data as eight bit values in consecutive storage locations. The operand of this directive may be either an expression or a string of character data. The expression must be able to be represented by an eight bit value. A text string may contain up to a maximum of sixteen char-

acters and will be stored as the numerical code equivalent of the individual characters in succession. The DW directive stores data as a sixteen bit address in two bytes, least significant byte first. The operand may be either an expression or a text string. The expression must be able to be represented by a sixteen bit value. The text string may contain up to sixteen characters.

Memory reservation. The DW (Define Storage) directives reserves a number of successive bytes for data storage. The operand is an expression, the value of which determines the number of bytes reserved. The contents of the spaces are unchanged by the operation and are not predictable without specific initialization.

Assembler termination. The END directive identifies the end of the assembly language program. It causes each pass of the assembler to terminate.

Symbol definition. The EQU (EQUal) directive assigns a value to a label. The expression in the operand field is evaluated and the resulting value assigned to the label preceding the directive.

Location counter control. The ORG (ORiGin) directive sets the location counter to the value of the expression in its operand field.

Output. The output of the assembler is hexadecimal object code and its associated hexadecimal location counter for each input line. The output is of the form:

location counter code text of source line

The symbol table, if requested, is presented prior to the assembled output. The preprocessor then reformats the assembler output into card images compatible with the CDL simulator loading subroutines and

initiates the loading process. The card image data is loaded directly into the simulator's storage area by the program. The card image output is also printed for reference.

Error Handling

The design philosophy for error handling in the preprocessor is to process the maximum amount of information possible in spite of recognized errors, without propagating those errors. Thus errors in the assembly option cards normally cause termination of assembly after the printing of the appropriate error message, because the effects of the errors on subsequent processing is unknown. The exception is an option with a default value, such as the listing and origin options, where recovery is made by assuming the default value. Within separate passes, processing is controlled by an error count. Pass one results are printed and assembly terminated only if there have been errors in the format of the program. Pass two errors are generally syntax errors, which result in the printing of the appropriate error messages with the output and a termination of the preprocessor reformatting and loading sequence. The net effect of the design philosophy is to flag as many errors as possible in a single run of the simulator, thus minimizing the total number of runs necessary to correct a program. This efficient interface to the operator is a visible benefit of the preprocessor.

APPENDIX A-2

ERROR CODES

Errors encountered in generation or running of the simulator are identified by a seven character code. This appendix lists these diagnostic codes and their associated meanings.

TRANSLATION ERRORS

CDL1001	MISSING HEADING STATEMENT
CDL1002	INVALID STATEMENT CONTINUATION
CDL1003	UNRECOGNIZED STATEMENT
CDL1004	EXCESSIVE STATEMENT LENGTH (MORE THAN 250 TERMS)
CDL1101	SYNTAX ERROR IN *HEADING STATEMENT
CDL1201	UNRECOGNIZED DEVICE DECLARATION
CDL1301	MISSING COMMA IN REGISTER DECLARATION
CDL1302	INVALID VARIABLE NAME IN REGISTER DECLARATION
CDL1303	INVALID REGISTER SIZE FORMAT
CDL1401	MISSING COMMA IN SUBREGISTER DECLARATION
CDL1402	GENERAL SYNTAX ERROR IN SUBREGISTER DECLARATION
CDL1403	UNDECLARED OR INVALID DEVICE REFERENCE IN SUBREGISTER DECLARATION
CDL1404	INVALID REFERENCE REGISTER BIT STRING DEFINITION IN SUB-REGISTER DECLARATION
CDL1501	MISSING COMMA IN MEMORY DECLARATION
CDL1502	GENERAL SYNTAX ERROR IN MEMORY
CDL1503	INVALID BIT STRING DESIGNATION IN MEMORY DECLARATION
CDL1504	UNDECLARED REGISTER OR INVALID DEVICE TYPE IN MEMORY DECLARATION
CDL1551	MISSING COMMA IN BUS DECLARATION
CDL1552	INVALID VARIABLE NAME IN BUS DECLARATION
CDL1553	INVALID BUS SIZE FORMAT
CDL1601	MISSING COMMA IN DECODER DECLARATION
CDL1602	GENERAL SYNTAX ERROR IN DECODER DECLARATION
CDL1603	UNDECLARED OR INVALID DEVICE NAME IN DECODERDECLARATION
CDL1701	SYNTAX ERROR IN CLOCK DECLARATION
CDL1702	TWO CLOCK DECLARATIONS
CDL1801	MISSING COMMA IN SWITCH DECLARATION
CDL1802	SYNTAX ERROR IN SWITCH DECLARATION
CDL1851	MISSING COMMA IN LIGHT DECLARATION
CDL1852	SYNTAX ERROR IN LIGHT DECLARATION
CDL1901	SYNTAX ERROR IN TERMINAL DECLARATION
CDL2001	SYNTAX ERROR IN BLOCK DECLARATION
CDL2101	SYNTAX ERROR IN 'DO' STATEMENT
CDL2102	INVALID OR UNDECLARED DEVICE NAME
CDL2103	SYNTAX ERROR IN CONDITIONAL MICROSTATEMENT
CDL2104	UNDECLARED OR INVALID DEVICE REFERENCE IN MICROSTATEMENT
CDL2105	INVALID USE OF CONSTANT
CDL2201	A BLANK LABEL MAY APPEAR ONLY IN AN 'OPERATOR' OR 'SEQUENCE' PROGRAM WITH ONE OR TWO ARGUMENTS
CDL2202	SYNTAX ERROR IN LABEL STATEMENT
CDL2203	UNDECLARED OR INVALID DEVICE NAME IN LABEL STATEMENT
CDL2301	SYNTAX ERROR IN EXPRESSION
CDL2401	SYNTAX ERROR IN DECODING EXPRESSION
CDL2402	INVALID OR UNDECLARED DEVICE REFERENCE
CDL2501	INVALID OR UNDECLARED DEVICE NAME
CDL2502	SYNTAX ERROR
CDL2601	SUBSCRIPT IS NOT A CONSTANT

CDL2602 LENGTH SPECIFIED EXCEEDS 72 BITS
CDL2603 SYNTAX ERROR IN SUBSCRIPT

SIMULATION ERRORS

CDL5001 INVALID CONTROL STATEMENT
STATEMENT IS IGNORED, SIMULATION CONTINUES
CDL5101 SYNTAX ERROR IN '*OUTPUT' STATEMENT
CDL5102 INVALID OR UNDECLARED DEVICE NAME IN OUTPUT LIST
NAME IS IGNORED, SIMULATION CONTINUES
CDL5201 SYNTAX ERROR IN '*LOAD' STATEMENT
CDL5202 INVALID OR UNDECLARED DEVICE IN '*LOAD' STATEMENT
CDL5301 MAXIMUM LABEL CYCLES TO BE SIMULATED NOT SPECIFIED
100 ASSUMED, SIMULATION CONTINUES
CDL5302 LABEL REPITITION COUNT NOT SPECIFIED IN '*SIM' STATEMENT
3 ASSUMED, SIMULATION CONTINUES
CDL5303 AMBIGUOUS LABEL EXPRESSION
CDL5304 ERROR IN MICROSTATEMENT
CDL5305 ERROR IN LABEL EXPRESSION
CDL5306 ERROR IN SWITCH LABEL EXPRESSION
CDL5401 UNDEFINED OPERATOR ENCOUNTERED DURING SIMULATION
CDL5402 VARIABLE LENGTH OF MORE THAN 64 BITS
CDL5403 MEMORY ADDRESSING ERROR
CDL5404 INVALID STORE REFERENCE OR COMPLEMENT
CDL5405 INVALID USE OF STANDARD LOGICAL OPERATOR
CDL5406 INVALID OR UNDEFINED OPERATOR
CDL5407 INVALID USE OF SUBSCRIPT
CDL5408 INVALID PARALLEL SEQUENCE CALL
CDL5409 INVALID CONDITIONAL TRANSFER
CDL5410 ERROR IN SEQUENCE TRANSLATION
CDL5501 OPERATOR LISTED IS INVALID
CDL5601 INVALID STORE REFERENCE
CDL5602 INVALID OR EXCESSIVE LENGTH OF VARIABLE TO BE STORED
CDL5701 SYNTAX ERROR IN 'SWITCH' STATEMENT

APPENDIX B

SIMULATION OF INTEL 8080 MICROPROCESSOR

```
$TRANSLATE
*MAIN
C ******************************************************************
C **                                                              **
C **          SIMULATION OF INTEL 8080 MICROPROCESSOR             **
C **                                                              **
C ******************************************************************
C
C                          HARDWARE SETUP
C
 REGISTER, W(7-0),Z(7-0),B(7-0),C(7-0),D(7-0),E(7-0),H(7-0),L(7-0),
1          SP(15-0),PC(15-0),ALATCH(15-0),ADDBUFFER(15-0),IR(7-0),
1          DATABUF(7-0),TEMP(7-0),A(7-0),ACLATCH(7-0),
1          CARRY,ZERO,SIGN,PARITY,CY1
 BUS,      INTERNAL(7-0)
C
C                      CONTROL SECTION HARDWARE
C
 REGISTER, SYNC,ORIN,READY,WAIT,NWR,HOLD,HLDA,INEN,INT,RESET,MEMR,
1          FLAGS,NWO,ST1,INP,OUT,HLTA,STACK,INTA,STSTR,
1          X(0-2),Y(0-2)
 REGISTER, MREF,MR1,MR2,MW1
 SWITCH,   INTS(OFF,ON),SWINT(OFF,ON)
 DECODER,  T(0-5)=X,M(0-5)=Y,
1          OP1(0-7)=IR(7-5),OP2(0-3)=IR(4-3),OP3(0-7)=IR(2-0),
1          OPD(0-7)=IR(5-3)
 TERMINAL, OF1=A(7),OF2=TEMP(7),OF3=A(7).ADD.TEMP(7),
1          OF4=A(3),OF5=TEMP(3),OF6=A(3).ADD.TEMP(3),
1          OF7=ALATCH(15),OF8=H(7),OF9=ALATCH(15).ADD.H(7)
 BLOCK,    SEVAL (IF (OP3(0)) THEN (TEMP=B),
1                 IF (OP3(1)) THEN (TEMP=C),
1                 IF (OP3(2)) THEN (TEMP=D),
1                 IF (OP3(3)) THEN (TEMP=E),
1                 IF (OP3(4)) THEN (TEMP=H),
1                 IF (OP3(5)) THEN (TEMP=L),
1                 IF (OP3(6)) THEN (MREF=1),
1                 IF (OP3(7)) THEN (TEMP=A))
 BLOCK,    DEVAL (IF(OPD(0)) THEN (B=TEMP),
1                 IF(OPD(1)) THEN (C=TEMP),
1                 IF(OPD(2)) THEN (D=TEMP),
1                 IF(OPD(3)) THEN (E=TEMP),
1                 IF(OPD(4)) THEN (H=TEMP),
1                 IF(OPD(5)) THEN (L=TEMP),
1                 IF(OPD(6)) THEN (MW1=1,ALAT=H-L,X=0,Y=2)ELSE(X=0,Y=1),
1                 IF(OPD(7)) THEN (A=TEMP))
 BUS,          ADMEM(15-0)
 MEMORY,       MEM(ADMEM)=MEM(0-3072,7-0)
 BUS,          ADPO(7-0)
 MEMORY,       PORT(ADPO)=PORT(0-100,7-0)
 CLOCK,        P(1)
C
C  INTEL 8228 SYSTEM CONTROLLER AND BUS DRIVER HARDWARE
C
 REGISTER, STAT(7-0),NMR,NMW,NIOR,NIOW,NINT
C
C  INTEL 8255 PROGRAMMABLE INTERFACE HARDWARE CONFIGURATION
C
 REGISTER,  INT1,INT2,INT3,INTR,RSET,STBA,ACKA,STBB,
1           APORT(7-0),BPORT(7-0),CPORT(7-0),CWORD(7-0),INTR,
1           IBFA,OBFA,IBFB
C
```

```
 TERMINAL,      SEL0=ADP0(0),SEL1=ADP0(1),CSL0=ADP0(2),RDL0=NIOR,
1               WRL0=NIOW
 DECODER,       CSET(0-7)=CWORD(3-1)
 SWITCH,    INIT(OFF,ON),INDATA(OFF,ON)
 BLOCK,    SETC   (IF (CSET(0)) THEN (CPORT(0)=1,INTB=1),
1                  IF (CSET(1)) THEN (CPORT(1)=1,IBFB=1),
1                  IF (CSET(2)) THEN (CPORT(2)=1,INT3=1),
1                  IF (CSET(3)) THEN (CPORT(3)=1,INTR=1),
1                  IF (CSET(4)) THEN (CPORT(4)=1,INT2=1),
1                  IF (CSET(5)) THEN (CPORT(5)=1,IBFA=1),
1                  IF (CSET(6)) THEN (CPORT(6)=1,INT1=1),
1                  IF (CSET(7)) THEN (CPORT(7)=1,OBFA=1))
C
 BLOCK,   RETC ( IF (CSET(0)) THEN (CPORT(0)=1,INTB=0),
1                  IF (CSET(1)) THEN (CPORT(1)=0,IBFB=0),
1                  IF (CSET(2)) THEN (CPORT(2)=0,INT3=0),
1                  IF (CSET(3)) THEN (CPORT(3)=0,INTR=0),
1                  IF (CSET(4)) THEN (CPORT(4)=0,INT2=0),
1                  IF (CSET(5)) THEN (CPORT(5)=0,IBFA=0),
1                  IF (CSET(6)) THEN (CPORT(6)=0,INT1=0),
1                  IF (CSET(7)) THEN (CPORT(7)=0,OBFA=0))
C
 BUS,    NUMBER(2-0)
 MEMORY,     INPUT(NUMB)=INPUT(0-7,7-0)
C   ***************************************************************************
C   ***                                                                     ***
C   ***                      CONTROL FUNCTIONS                              ***
C   ***                                                                     ***
C   ***************************************************************************
C
C   8080 CONTROL FUNCTIONS
C
 /RESET*P(0)/ HLTA=0,WAIT=0,READY=1,PC=0,X=6,Y=6
C
 /SYNC*P(0)/   DATABUF=MEMR-INP-M(1)-OUT-HLTA-STACK-NWO-INTA,STSTR=0,
           SYNC=0
C
C   8228 CONTROL FUNCTIONS
C
 /STSTR'*P(1)/ STAT=DATABUF,NMR=(DBIN*MEMR)',NMW=(NWR'*OUT')',
           NIOR=(DBIN*INP)',NIOW=(NWR'*OUT)',NINT=(DBIN*INTA)',STSTR=1
C
 /M(1)*T(3)*P(1)*READY*INTA/ IR=:FF
C
C   ***************************************************************************
C   **                                                                       **
C   **                      UTILITY ROUTINES                                 **
C   **                                                                       **
C   ***************************************************************************
C
C         START SWITCH AND EXTERNAL INTERRUPT SWITCH
C
 /INTST(ON)/ A=0,READY=1,X=6,Y=6,NWO=1,NWR=1,NMR=1,NMW=1,NIOR=1,NIOW=1,
           NINT=1,STSTR=1
 /SWINT(ON)/ INT=1,SWINT=OFF
C
C         MEMORY READ CYCLE
C
 /DBIN*NMR'*P(0)/ ADMEM=ADDBUFFER
 /DBIN*NMR'*P(1)/ DATAB=MEM(ADME),READY=1,DBIN=0,MEMR=0,NMR=1
C
```

```
 /M(3)*T(1)*P(1)*READY*MR2/ ADDBUFFER=ALATCH,
           SYNC=1,DBIN=1,READY=0,MEMR=1
 /M(3)*T(2)*P(1)*READY*MR2/ MR2=0
C
C        WRITE  BYTES INTO MEMORY
C
 /T(1)*P(1)*MWI*READY/ ADDBUFFER=ALATCH,INTERNAL=TEMP,
           SYNC=1,MWO=0,READY=0,MWR=0
 /T(2)*P(1)*MWI*READY/ MWI=0
C
C ***************************************************************************
C
C             8255 CONTROL FUNCTIONS
C
C ***************************************************************************
C
C              SWITCHING OPERATIONS
C
 /INITION)/ ADPO(2)=1,CPORT(2)=1,CPORT(4)=1,CPORT(6)=1,STBA=1,
           ACKA=1,STBB=1,INIT=OFF
 /INDATA(ON)/ STBA=0,APORT=INPU(NUMBER),NUMBER=NUMBER.COUNT.,
           PORT(0)=INPU(NUMBER),INDATA=OFF
C
C              RESET
C
 /RSET*P(1)/ CWORD=0,INT1=0,INT2=0,INT3=0
C
C              STROBE ACTION
C
 /STBA'*P(1)/ CPORT(4)=0,IBFA=1,CPORT(5)=1
 /STBB'*P(1)/ CPORT(2)=0,IBFB=1,CPORT(1)=1
 /STBA'*IBFA*P(0)/ STBA=1
 /STBB'*IBFB*P(0)/ STBB=1
 /ACKA'*OBFA'*P(0)/ ACKA=1,CPORT(6)=0
C
C        INTERRUPT LOGIC
C
 /(IBFA*STBA'*INT2*RDLO)+(OBFA'*ACKA'*INT1*WRLO)*P(0)/CPORT(3)=1,INTR=1
 /IBFB*STBB'*INT3*RDLO*P(0)/ CPORT(0)=1,INTR=1
 /(INTR*INTB)*P(1)/  INT=1
 /(INTR*INTB)*INT*P(0)/ INTR=0,INTB=0
C
C      BIT SET/RESET FUNCTION
C
 /CSLO'*(CWORD(7)')*CWORD(0)*SELO*SEL1*P(0)/ DO/SETC
 /CSLO'*(CWORD(7)')*(CWORD(0)')*SELO*SEL1*P(0)/ DO/RETC
C
C      OUTPUT TO 8255
C
 /CSLO'*WRLO'*SELO*SEL1*P(1)/ CWORD=DATAB
 /CSLO'*WRLO'*SELO'*SEL1'*P(1)/ APORT=DATAB
 /CSLO'*WRLO'*SELO*SEL1'*P(1)/ BPORT=DATAB
 /CSLO'*WRLO'*SELO'*SEL1*P(1)/ CPORT=DATAB
C
C      INPUT FROM 8255
C
 /CSLO'*RDLO'*SELO'*SEL1'*P(1)/ DATAB=APORT
 /CSLO'*RDLO'*SELO*SEL1'*P(1)/ DATAB=BPORT
 /CSLO'*RDLO'*SELO'*SEL1*P(1)/ DATAB=CPORT
C
C      INPUT AND OUTPUT TERMINATION
```

```
C
 /CSLO**P(0)/ ADPO(2)=1
C
C  ***************************************************************************
C  **                                                                       **
C  **                    INTEL 8080 INSTRUCTION SET                         **
C  **                                                                       **
C  ***************************************************************************
C
C          ADI   ACI   SUI   SBI   ANI   XRI   ORI   CPI   MVI
C
 /M(1)*T(4)*P(1)*READY*(IR(7).ERA.IR(6))**OP3(6)/ ALATCH=PC,
           MR[=1,X=0,Y=2
 /M(2)*T(2)*P(1)*READY*(IR(7).ERA.IR(6))**OP3(6)/ MR1=0
 /M(2)*T(3)*P(1)*READY*(IR(7).ERA.IR(6))**OP3(6)/ PC=ALATCH,
           TEMP=DATABUF,IR(6)=IR(6)*,X-4,Y=1
C
C          RLC   RRC   RAL   RAR
C
 /M(1)*T(4)*P(1)*READY*OP1(0)*OP3(7)/   X=0,Y=1
 /M(1)*T(4)*P(1)*READY*OP1(0)*OP2(0)*OP3(7)/ A=A(6-0)-A(7),CARRY=A(7)
 /M(1)*T(4)*P(1)*READY*OP1(0)*OP2(1)*OP3(7)/ A=A(0)-A(7-1),CARRY=A(0)
 /M(1)*T(4)*P(1)*READY*OP1(0)*OP2(2)*OP3(7)/ A=A(6-0)-CARRY,CARRY=A(7)
 /M(1)*T(4)*P(1)*READY*OP1(0)*OP2(3)*OP3(7)/ A=CARRY-A(7-1),CARRY=A(0)
C
C          DAA   CMA   STC   CMC
C
 /M(1)*T(4)*P(1)*READY*OP1(1)*OP2(0)*OP3(7)/ IF (CY) THEN
           (A=A.ADD.6,INTE=A.ADD.6) ELSE (IF (A(3-0).GE.10) THEN
           (A=A.ADD.6,INTE=A.ADD.6))
 /M(1)*T(5)*P(1)*READY*OP1(1)*OP2(0)*OP3(7)/ FLAGS=1,X=0,Y=1,
           IF (CARRY.EQ.1) THEN (A=A.ADD.96,INTE=A.ADD.96) ELSE
          (IF (A(7-4).GE.10) THEN (A=A.ADD.96,CARRY=1,INTE=A.ADD.96)ELSE
          (IF (A(7-4).EQ.9) THEN(IF (A(3-0).GE.10) THEN
          (A=A.ADD.96,CARRY=1,INTE=A.ADD.96))))
 /M(1)=T(4)*P(1)*READY*OP1(1)*OP2(0)**OP3(7)/ X=0,Y=1
 /M(1)*T(4)*P(1)*READY*OP1(1)*OP2(1)*OP3(7)/ A=A*
 /M(1)*T(4)*P(1)*READY*OP1(1)*OP2(2)*OP3(7)/ CARRY=1
 /M(1)*T(4)*P(1)*READY*OP1(1)*OP2(3)*OP3(7)/ CARRY=CARRY*
C
C          ANA   XRA   ORA   CMP
C
 /M(1)*T(4)*P(1)*READY*OP1(5)/   DC/SEVAL
 /M(1)*T(5)*P(1)*READY*OP1(5)*MREF / X=0,Y=2
 /M(1)*T(5)*P(1)*READY*OP1(5)*MREF*/ CARRY=0,X=0,Y=1,
 /M(1)*T(5)*P(1)*READY*OP1(5)*OP2(0)*MREF*/ A=A*TEMP
 /M(1)*T(5)*P(1)*READY*OP1(5)*OP2(1)*MREF*/ A=A.ERA.TEMP
 /M(1)*T(5)*P(1)*READY*OP1(5)*OP2(2)*MREF*/ A=A+TEMP
 /M(1)*T(5)*P(1)*READY*OP1(5)*OP2(3)*MREF*/ IF (A.EQ.TEMP) THEN (ZERO=1)
          , IF (A.LT.TEMP) THEN (CARRY=1)
 /M(3)*T(1)*P(1)*READY*OP1(5)/ X=4,Y=1
C
C          ADD   ADC   SUB   SBB
C
 /M(1)*T(4)*P(1)*READY*OP1(4)/   DC/SEVAL
 /M(1)*T(5)*P(1)*READY*OP1(4)*MREF / X=0,Y=2
 /M(1)*T(5)*P(1)*READY*OP1(4)*MREF*/ X=0,Y=1,FLAGS=1
 /M(1)*T(5)*P(1)*READY*OP1(4)*OP2(0)*MREF*/ A=A.ADD.TEMP,
           IF ((OF1*OF2*OF3*)+(OF1**OF2**OF3)) THEN (CARRY=1) ELSE
           (CARRY=0),INTEREAL=A.ADD.TEMP,
           IF ((OF4*OF5*OF6*)+(OF4**OF5**OF6)) THEN (CY1=1) ELSE (CY1=0)
```

```
/M(1)*T(5)*P(1)*READY*OP1(4)*OP2(1)*MREF*/ A=A.ADD.TEMP.ADD.CARRY,
          IF ((A.ADD.TEMP).EQ.:F) THEN (CARRY=1) ELSE
        (IF ((OF1*OF2*OF3')+(OF1'*OF2'*OF3)) THEN (CARRY=1) ELSE
        (CARRY=0)),INTERNAL=A.ADD.TEMP.ADD.CARRY,
          IF ((A(3-0).ADD.TEMP(3-0)).EQ.:F) THEN (CY1=1) ELSE
        (IF ((OF4*OF5*OF6')+(OF4'*OF5'*OF6)) THEN (CY1=1)ELSE (CY1=0))
/M(1)*T(5)*P(1)*READY*OP1(4)*OP2(2)*MREF*/ A=A.SUB.TEMP,
          INTERNAL=A.SUB.TEMP,
          IF (A.LT.TEMP) THEN (CARRY=1) ELSE (CARRY=0),
          IF (A(3-0).LT.TEMP(3-0)) THEN (CY1=1) ELSE (CY1=0)
/M(1)*T(5)*P(1)*READY*OP1(4)*OP2(3)*MREF*/ A=A.SUB.TEMP.SUB.CARRY,
          INTERNAL=A.SUB.TEMP.SUB.CARRY,
          IF (A.LT.(TEMP.SUB.CARRY)) THEN (CARRY=1) ELSE (CARRY=0),
          IF (A(3-0).LT.(TEMP(3-0).SUB.CARRY)) THEN (CY1=1)ELSE(CY1=0)
/M(3)*T(1)*P(1)*READY*OP1(4)/  X=4,Y=1
C
C        JMP   JNZ   JZ   JNC   JC   JPO   JPE   JM   JP
C      (CALL   CNZ   CZ   CNC   CC   CPO   CPE   CM   CP
C       RET)  RNZ   RZ   RNC   RC   RPO   RPE   RM   RP
C
/M(1)*T(4)*P(1)*READY*OP1(6)*OP2(0)*OP3(3)/ X=0,Y=2,IR(0)=0,ALATCH=PC,
          MR2=1
/M(1)*T(4)*P(1)*READY*OP1(6)*OP2(1)*(OP3(5)+OP3(1))/ X=0,Y=2,IR(0)=0,
          MR2=1,IF (OP3(5)) THEN (ALATCH=PC) ELSE (ALATCH=SP,STACK=1)
/M(1)*T(4)*P(1)*READY*OP1(6)*OP2(0)*(OP3(4)+OP3(2)+OP3(0))/ IF (ZERO')
      THEN (X=0,Y=2,MR2=1,IF(OP3(0))THEN(ALATCH=SP,STACK=1)
      ELSE (ALATCH=PC)) ELSE (X=0,Y=1, IF (OP3(0)') THEN (PC=PC.ADD.2))
/M(1)*T(4)*P(1)*READY*OP1(6)*OP2(1)*(OP3(4)+OP3(2)+OP3(0))/ IF (ZERO)
      THEN (X=0,Y=2,MR2=1,IF(OP3(0))THEN(ALATCH=SP,STACK=1)
      ELSE (ALATCH=PC)) ELSE (X=0,Y=1, IF (OP3(0)') THEN (PC=PC.ADD.2))
/M(1)*T(4)*P(1)*READY*OP1(6)*OP2(2)*(OP3(4)+OP3(2)+OP3(0))/ IF (CARRY')
      THEN (X=0,Y=2,MR2=1,IF(OP3(0))THEN(ALATCH=SP,STACK=1)
      ELSE (ALATCH=PC)) ELSE (X=0,Y=1, IF (OP3(0)') THEN (PC=PC.ADD.2))
/M(1)*T(4)*P(1)*READY*OP1(6)*OP2(3)*(OP3(4)+OP3(2)+OP3(0))/ IF (CARRY)
      THEN (X=0,Y=2,MR2=1,IF(OP3(0))THEN(ALATCH=SP,STACK=1)
      ELSE (ALATCH=PC)) ELSE (X=0,Y=1, IF (OP3(0)') THEN (PC=PC.ADD.2))
/M(1)*T(4)*P(1)*READY*OP1(7)*OP2(0)*(OP3(4)+OP3(2)+OP3(0))/ IF (PARITY')
      THEN (X=0,Y=2,MR2=1,IF(OP3(0))THEN(ALATCH=SP,STACK=1)
      ELSE (ALATCH=PC)) ELSE (X=0,Y=1, IF (OP3(0)') THEN (PC=PC.ADD.2))
/M(1)*T(4)*P(1)*READY*OP1(7)*OP2(1)*(OP3(4)+OP3(2)+OP3(0))/ IF (PARITY)
      THEN (X=0,Y=2,MR2=1,IF(OP3(0))THEN(ALATCH=SP,STACK=1)
      ELSE (ALATCH=PC)) ELSE (X=0,Y=1, IF (OP3(0)') THEN (PC=PC.ADD.2))
/M(1)*T(4)*P(1)*READY*OP1(7)*OP2(2)*(OP3(4)+OP3(2)+OP3(0))/ IF (SIGN')
      THEN (X=0,Y=2,MR2=1,IF(OP3(0))THEN(ALATCH=SP,STACK=1)
      ELSE (ALATCH=PC)) ELSE (X=0,Y=1, IF (OP3(0)') THEN (PC=PC.ADD.2))
/M(1)*T(4)*P(1)*READY*OP1(7)*OP2(3)*(OP3(4)+OP3(2)+OP3(0))/ IF (SIGN)
      THEN (X=0,Y=2,MR2=1,IF(OP3(0))THEN(ALATCH=SP,STACK=1)
      ELSE (ALATCH=PC)) ELSE (X=0,Y=1, IF (OP3(0)') THEN (PC=PC.ADD.2))
/M(3)*T(3)*P(1)*READY*IR(7)*IR(6)*(OP3(2)+OP3(0))/ PC.DATABUF=TEMP,
          X=0,Y=1,IF (OP3(0)) THEN (SP=SP.ADD.2,STACK=0)
/M(3)*T(3)*P(1)*READY*IR(7)*IR(6)*OP3(4)/ W.DATABUF,Z=TEMP,
          *LATCH=SP.SUB.1,TEMP=PC(15-8),X=0,Y=4,MWI=1,STACK=1
/M(4)*T(3)*P(1)*READY*IR(7)*IR(6)*OP3(4)/ ALATCH=ALATCH.SUB.1,
          TEMP=PC(7-0).ADD.2,X=0,Y=5,MWI=1
/M(5)*T(3)*P(1)*READY*IR(7)*IR(6)*OP3(4)/ PC.W=Z,SP=SP.SUB.2,X=3,Y=1,
          STACK=0
C
C        INX B   INX D   INX H   INX SP
C        DCX B   DCX D   DCX H   DCX SP
C
/M(1)*T(4)*P(1)*READY*OP1(0)*OP2(3)*OP3(3)/ ALATCH=ID(1:0),INT.
```

```
 /M(1)*T(4)*P(1)*READY*OP1(0)*OP2(2)*OP3(3)/ ALATCH=(D-E).COUNT.
 /M(1)*T(4)*P(1)*READY*OP1(1)*OP2(3)*OP3(3)/ ALATCH=(H-L).COUNT.
 /M(1)*T(4)*P(1)*READY*OP1(1)*OP2(2)*OP3(3)/ ALATCH=(SP).COUNT.
 /M(1)*T(4)*P(1)*READY*OP1(0)*OP2(1)*OP3(3)/ ALATCH=(B-C).SUB.1
 /M(1)*T(4)*P(1)*READY*OP1(0)*OP2(3)*OP3(3)/ ALATCH=(D-E).SUB.1
 /M(1)*T(4)*P(1)*READY*OP1(1)*OP2(1)*OP3(3)/ ALATCH=(H-L).SUB.1
 /M(1)*T(4)*P(1)*READY*OP1(1)*OP2(3)*OP3(3)/ ALATCH=SP.SUB.1
 /M(1)*T(5)*P(1)*READY*OP1(0)*(OP2(0)+OP2(1))*OP3(3)/ X=0,Y=1,
          B=ALATCH(15-8),C=ALATCH(7-0)
 /M(1)*T(5)*P(1)*READY*OP1(0)*(OP2(2)+OP2(3))*OP3(3)/ X=0,Y=1,
          D=ALATCH(15-8),E=ALATCH(7-0)
 /M(1)*T(5)*P(1)*READY*OP1(1)*(OP2(0)+OP2(1))*OP3(3)/ X=0,Y=1,
          H=ALATCH(15-8),L=ALATCH(7-0)
 /M(1)*T(5)*P(1)*READY*OP1(1)*(OP2(2)+OP2(3))*OP3(3)/ X=0,Y=1,
          SP=ALATCH
C
C         LXI B  LXI D  LXI H  LXI SP
C
 /M(1)*T(4)*P(1)*READY*(OP1(0)+OP1(1))*(OP2(0)+OP2(2))*OP3(1)/
          X=0,Y=2,ALATCH=PC,PC=PC.ADD.2,MR2=1
 /M(3)*T(3)*P(1)*READY*OP1(0)*OP2(0)*OP3(1)/ X=0,Y=1,C=TEMP,B=DATABUF
 /M(3)*T(3)*P(1)*READY*OP1(0)*OP2(2)*OP3(1)/ X=0,Y=1,E=TEMP,D=DATABUF
 /M(3)*T(3)*P(1)*READY*OP1(1)*OP2(0)*OP3(1)/ X=0,Y=1,L=TEMP,H=DATABUF
 /M(3)*T(3)*P(1)*READY*OP1(1)*OP2(2)*OP3(1)/ X=0,Y=1,SP=DATABUF+TEMP
C
C         STAX B  STAX D  LDAX B  LDAX D
C
 /MR1)*T(4)*P(1)*READY*OP1(0)*(OP2(0)+OP2(2))*OP3(2)/ X=0,Y=2,TEMP=A
 /M(1)*T(4)*P(1)*READY*OP1(0)*OP2(0)*OP3(2)/ ALATCH=B-C,MW1=1
 /M(1)*T(4)*P(1)*READY*OP1(0)*OP2(2)*OP3(2)/ ALATCH=D-E,MW1=1
 /M(1)*T(4)*P(1)*READY*OP1(0)*OP2(1)*OP3(2)/ ALATCH=B-C,
          MR1=1,X=0,Y=2
 /M(1)*T(4)*P(1)*READY*OP1(0)*OP2(3)*OP3(2)/ ALATCH=D-E,
          MR1=1,X=0,Y=2
 /M(2)*T(2)*P(1)*READY*OP1(0)*(OP2(1)+OP2(3))*OP3(2)/ MR1=0
 /M(2)*T(3)*P(1)*READY*OP1(0)*(OP2(1)+OP2(3))*OP3(2)/ A=DATABUF,X=0,Y=1
 /M(2)*T(3)*P(1)*READY*OP1(0)*(OP2(0)+OP2(2))*OP3(2)/ X=0,Y=1
C
C         DAD B  DAD D  DAD H  DAD SP
C
 /M(1)*T(4)*P(1)*READY*OP1(0)*OP2(1)*OP3(1)/ ALATCH=B-C
 /M(1)*T(4)*P(1)*READY*OP1(0)*OP2(3)*OP3(1)/ ALATCH=D-E
 /M(1)*T(4)*P(1)*READY*OP1(1)*OP2(1)*OP3(1)/ ALATCH=H-L
 /M(1)*T(4)*P(1)*READY*OP1(1)*OP2(3)*OP3(1)/ ALATCH=SP
 /M(1)*T(5)*P(1)*READY*(OP1(0)+OP1(1))*(OP2(1)+OP2(3))*OP3(1)/ X=3,Y=2,
          IF ((OF7*OF8*CL9*)+(OF7**OF8**OF9)) THEN (CARRY=1) ELSE
          (CARRY=0),ALATCH (H-L).ADD.ALATCH
 /M(2)*T(4)*P(1)*READY*(OP1(0)+OP1(1))*(OP2(1)+OP2(3))*OP3(1)/ X=0,Y=1,
          H=ALATCH(15-8),L=ALATCH(7-0),
C
C         MOV  HLT
C
 /M(1)*T(4)*P(1)*READY*IR(7)**IR(6)/ IF (OP1(3)*OP2(2)*OP3(6)) THEN
          (HLTA=1,X=0,Y=2) ELSE (MOVSEVAL)
 /M(1)*T(5)*P(1)*READY*IR(7)**IR(6)*MREF*/ DC/DEVAL
 /M(1)*T(5)*P(1)*READY*IR(7)**IR(6)*MREF / X=0,Y=2
 /M(2)*T(1)*P(1)*HLTA*READY/ SYNC=1,MEMR=1
 /M(2)*T(2)*P(1)*HLTA/ WAIT=1,READY=0
 /M(2)*T(2)*P(1)*READY*HLTA**IR(7)**IR(6)*OP3(6)/ MREF=0,TEMP=DATABUF
 /M(2)*T(3)*P(1)*READY*IR(7)**IR(6)*OP3(6)/ DC/DEVAL
 /M(2)*T(3)*P(1)*READY*IR(7)**IR(6)*OP3(6)/ X=0,Y=1
```

```
C
C         INR  DCR
C
 /M(1)*T(4)*P(1)*READY*IR(7)'*IR(6)'*OP3(4)/
            IF (OPD(0)) THEN (B=B.COUNT.,INTE=B.COUNT.),
            IF (OPD(1)) THEN (C=C.COUNT.,INTE=C.COUNT.),
            IF (OPD(2)) THEN (D=D.COUNT.,INTE=D.COUNT.),
            IF (OPD(3)) THEN (E=E.COUNT.,INTE=E.COUNT.),
            IF (OPD(4)) THEN (H=H.COUNT.,INTE=H.COUNT.),
            IF (OPD(5)) THEN (L=L.COUNT.,INTE=L.COUNT.),
            IF (OPD(6)) THEN (MREF=1,X=0,Y=2) ELSE (X=0,Y=1,FLAGS=1),
            IF (OPD(7)) THEN (A=A.COUNT.,INTE=A.COUNT.)
 /M(1)*T(4)*P(1)*READY*IR(7)'*IR(6)'*OP3(5)/
            IF (OPD(0)) THEN (B=B.SUB.1,INTE=B.SUB.1),
            IF (OPD(1)) THEN (C=C.SUB.1,INTE=C.SUB.1),
            IF (OPD(2)) THEN (D=D.SUB.1,INTE=D.SUB.1),
            IF (OPD(3)) THEN (E=E.SUB.1,INTE=E.SUB.1),
 /M(1)*T(4)*P(1)*READY*IR(7)'*IR(6)'*OP3(5)/
            IF (OPD(4)) THEN (H=H.SUB.1,INTE=H.SUB.1),
            IF (OPD(5)) THEN (L=L.SUB.1,INTE=L.SUB.1),
            IF (OPD(6)) THEN (MREF=1,X=0,Y=2) ELSE (X=0,Y=1,FLAGS=1),
            IF (OPD(7)) THEN (A=A.SUB.1,INTE=A.SUB.1)
 /M(2)*T(2)*P(1)*READY*IR(7)'*IR(6)'*(OP3(4)+OP3(5))/ MREF=0
 /M(2)*T(3)*P(1)*READY*IR(7)'*IR(6)'*(OP3(4)+OP3(5))/ X=0,Y=3,
            IF (OP3(4)) THEN (TEMP=DATABUF.COUNT.)
            ELSE (TEMP=DATABUF.SUB.1),MW1=1,ALATCH=H-L
 /M(3)*T(3)*P(1)*READY*IR(7)'*IR(6)'*(OP3(4)+OP3(5))/ X=0,Y=1,FLAGS=1
C
C         PUSH B  PUSH D  PUSH H  PUSH PSW
C
 /M(1)*T(4)*P(1)*READY*(OP1(6)+OP1(7))*(OP2(1)')*OP3(5)/ALATCH=SP.SUB.1,
            X=0,Y=2,MW1=1,STACK=1
 /M(1)*T(4)*P(1)*READY*OP1(6)*OP2(0)*OP3(5)/ TEMP=B
 /M(1)*T(4)*P(1)*READY*OP1(6)*OP2(2)*OP3(5)/ TEMP=D
 /M(1)*T(4)*P(1)*READY*OP1(7)*OP2(0)*OP3(5)/ TEMP=H
 /M(1)*T(4)*P(1)*READY*OP1(7)*OP2(2)*OP3(5)/ TEMP=A
 /M(2)*T(3)*P(1)*READY*(OP1(6)+OP1(7))*(OP2(1)')*OP3(5)/ALATCH=SP.SUB.2,
            X=0,Y=3,MW1=1
 /M(2)*T(3)*P(1)*READY*OP1(6)*OP2(0)*OP3(5)/ TEMP=C
 /M(2)*T(3)*P(1)*READY*OP1(6)*OP2(2)*OP3(5)/ TEMP=E
 /M(2)*T(3)*P(1)*READY*OP1(7)*OP2(0)*OP3(5)/ TEMP=L
 /M(2)*T(3)*P(1)*READY*OP1(7)*OP2(2)*OP3(5)/
            TEMP=CARRY-1-PARITY-0-CY1-0-ZERO-SIGN
 /M(3)*T(3)*P(1)*READY*(OP1(6)+OP1(7))*(OP2(1)')*OP3(5)/SP=SP.SUB.2,
            X=0,Y=1,STACK=0
C
C         POP B  POP D  POP H  POP PSW
C
 /M(1)*T(4)*P(1)*READY*(OP1(6)+OP1(7))*(OP2(0)+OP2(2))*OP3(1)/
            ALATCH=SP,X=0,Y=2,MW2=1,STACK=1
 /M(3)*T(3)*P(1)*READY*(OP1(6)+OP1(7))*(OP2(0)+OP2(2))*OP3(1)/
            SP=SP.ADD.2,X=0,Y=1,STACK=0
 /M(3)*T(3)*P(1)*READY*OP1(6)*OP2(0)*OP3(1)/ C=TEMP,B=DATABUF,
 /M(3)*T(3)*P(1)*READY*OP1(6)*OP2(2)*OP3(1)/ E=TEMP,D=DATABUF,
 /M(3)*T(3)*P(1)*READY*OP1(7)*OP2(0)*OP3(1)/ L=TEMP,H=DATABUF,
 /M(3)*T(3)*P(1)*READY*OP1(7)*OP2(2)*OP3(1)/ A=DATABUF,CARRY=TEMP(7),
            PARITY=TEMP(5),CY1=TEMP(3),ZERO=TEMP(1),SIGN=TEMP(0)
C
C        .STA  LDA  SHLD  LHLD
C
 /M(1)*T(4)*P(1)*READY*OP1(1)*OP3(2)/ X=2,Y=2,ALATCH=WZ,IF WZ.ADD.2,
```

```
            MR2=1
 /M(3)*T(3)*P(1)*READY*OP1(1)*OP2(0)*OP3(2)/ TEMP=L,ALATCH=DATABUF-TEMP,
            X=0,Y=4,MW1=1
 /M(3)*T(3)*P(1)*READY*OP1(1)*OP2(2)*OP3(2)/ TEMP=A,X=0,Y=4,
            ALATCH=DATABUF-TEMP,MW1=1
 /M(4)*T(3)*P(1)*READY*OP1(1)*OP2(2)*OP3(2)/ X=0,Y=1
 /M(3)*T(3)*P(1)*READY*OP1(1)*(OP2(1)+OP2(3))*OP3(2)/ X=0,Y=4
 /M(4)*T(1)*P(1)*READY*OP1(1)*(OP2(1)+OP2(3))*OP3(2)/ X=2,Y=4,
            ADDRBUFFER=DATABUF-TEMP,SYNC=1,DBIN=1,MEMR=1,READY=0
 /M(4)*T(3)*P(1)*READY*OP1(1)*OP2(0)*OP3(2)/ TEMP=H,ALATCH=ALAT.COUNT.,
            X=0,Y=5,MW1=1
 /M(4)*T(3)*P(1)*READY*OP1(1)*OP2(1)*OP3(2)/ X=0,Y=5
 /M(5)*T(1)*P(1)*READY*OP1(1)*OP2(1)*OP3(2)/ L=DATABUF,X=2,Y=5,
            ADDRBUFFER=ADDRBUFFER.COUNT.,SYNC=1,READY=0,DBIN=1,MEMR=1
 /M(4)*T(3)*P(1)*READY*OP1(1)*OP2(3)*OP3(2)/ A=DATABUF,X=0,Y=1
 /M(5)*T(3)*P(1)*READY*OP1(1)*OP2(0)*OP3(2)/ X=0,Y=1
 /M(5)*T(3)*P(1)*READY*OP1(1)*OP2(1)*OP3(2)/ H=DATABUF,X=0,Y=1
C
C           DI   EI   NOP
C
 /M(1)*T(4)*P(1)*READY*OP1(7)*OP2(2)*OP3(3)/ INEN=0,X=0,Y=1
 /M(1)*T(4)*P(1)*READY*OP1(7)*OP2(3)*OP3(3)/ INEN=1,X=0,Y=1
 /M(1)*T(4)*P(1)*READY*OP1(0)*OP2(0)*OP3(0)/ X=0,Y=1
C
C           PCHL   SPHL   XTHL   xCHG
C
 /M(1)*T(4)*P(1)*READY*OP1(7)*OP2(1)*OP3(1)/ PC=H-L,X=0,Y=1
 /M(1)*T(4)*P(1)*READY*OP1(7)*OP2(3)*OP3(1)/ SP=H-L,X=0,Y=1
 /M(1)*T(4)*P(1)*READY*OP1(7)*OP2(1)*OP3(3)/ D=D,D=H,E=L,L=E,X=0,Y=1
 /M(1)*T(4)*P(1)*READY*OP1(7)*OP2(0)*OP3(3)/ ALATCH=SP,X=0,Y=2,MR2=1,
            STACK=1
 /M(3)*T(3)*P(1)*READY*OP1(7)*OP2(0)*OP3(3)/ L=TEMP,H=DATABUF,TEMP=H,
            W=L,X=0,Y=4,MW1=1
 /M(4)*T(3)*P(1)*READY*OP1(7)*OP2(0)*OP3(3)/ TEMP=W,ALATCH=ADDBUF.SUB.1,
            X=0,Y=5,MW1=1
 /M(5)*T(3)*P(1)*READY*OP1(7)*OP2(0)*OP3(3)/ X=0,Y=1,STACK=0
C
C           RST
C
 /M(1)*T(4)*P(1)*READY*IR(7)*IR(6)*OP3(7)/  X=0,Y=2,TEMP=PC(15-8),
            ALATCH=SP.SUB.1,MW1=1,STACK=1
 /M(2)*T(3)*P(1)*READY*IR(7)*IR(6)*OP3(7)/  X=0,Y=3,TEMP=PC(7-0),
            ALATCH=ALATCH.SUB.1,MW1=1
 /M(3)*T(3)*P(1)*READY*IR(7)*IR(6)*OP3(7)/  X=0,Y=1,SP.SP.SUB.2,STACK=0,
            PC=:0-:0-:0-:0-:0-:0-:0 :0-:0 :0-IR(5) IR(4) IR(3) :0-:0-:0
C
C           OUT   IN
C
 /M(1)*T(4)*P(1)*READY*OP1(6)*(OP2(2)+OP2(3))*OP3(3)/ ALATCH PC,
            PC=PC.COUNT.,X=0,Y=2,MR1=1
 /M(3)*T(1)*P(1)*READY*OP1(6)*OP2(2)*OP3(3)/ ADDRBUFFER TEMP=TEMP,OUT=1,
            INTE=A,SYNC=1,READY=0,WRO=0,MWR=0
 /M(3)*T(1)*P(1)*READY*OP1(6)*OP2(3)*OP3(3)/ ADDRBUFFER TEMP=TEMP,INP=1,
            SYNC=1,CRTL=1,READY=0
 /M(3)*T(2)*P(1)*READY*OP1(6)*(OP2(2)+OP2(3))*OP3(3)/ X=2,Y=3
 /M(2)*T(3)*P(1)*READY*OP1(6)*OP2(2)*OP3(3)/ X=0,Y=1
 /M(4)*T(3)*P(1)*READY*OP1(6)*OP2(3)*OP3(3)/ A=DATABUF,X=0,Y=1
 END
C ***************************************************************
C **                                                           **
C **      END OF TRANSLATION, BEGIN SIMULATION                 **
```

```
C **                                                                **
C ****************************************************************
$SIMULATE
*OUTPUT    LABEL(1,2)=Y,X,A,DATABUF,ADDBUFFER,INEN,INT,
          IR,PC,TEMP,STAT,CWORD,CPORT,APORT,INT1,INT2,INTR
*SWITCH     1,INIST=ON
*SWITCH     2,INIT=ON
*SWITCH     75,INDATA=ON
*LOAD
 ASM 8080      MEM
 LIST
 NORG
           DW        00C1H              ;STORED DATA
           ORG       56
*               INTERRUPT HANDLER ROUTINE
 INPUT:    PUSH      PSW                ;SAVE A
           MVI       A,06               ;KILL INTR SIGNAL
           OUT       3
           MVI       A,0BH              ;KILL IBFA
           OUT       3
           IN        0                  ;RETRIEVE INPUT CHARACTER
           STA       0201H              ;STORE IT
           POP       PSW                ;RESTORE A
           EI                           ;REENABLE INTERRUPT
           RET                          ;RETURN
           ORG       0100H
*                 MAIN ROUTINE
 SETUP:    EI                           ;ENABLE INTERRUPT
           MVI       A,09               ;SET INT2
           OUT       3
 OUTPUT:   LDA       0200H              ;LOAD CHARACTER
           OUT       0                  ;OUTPUT TO PORT A
           HLT
           ORG       0200H
           DW        00C2H              ;STORED INFO
           END
 PC=:0100
 INPUT(:0)=:C1
*SIM         800,3
```

APPENDIX C

PREPROCESSOR FORTRAN ROUTINES

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCC       FORTRAN SUBRCUTINE,  TITLE : ASMINT         CCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C   DRIVER ROUTINE FOR CROSS ASSEMBLER AND LOADER ROUTINES
C
C
      SUBROUTINE ASMINT (A,JN,J1,J2,*)
      INTEGER H33,H34,H45,H46,H47,H48,H49,H73
      DATA H33,   H34,   H45,   H46,   H47,   H48,   H49
     C     /3HASM, 3HORG, 4HLIST,4HNONE,4HNORG,4H8080,4H
C
      COMMON /TABLES/ SU(7,15),SL(3,15),LB(4,250),CL(4),
     *          NSU,NSL,NLB,NCL,NP,NST,NSY,NPSAVE,MAXP,P(1)
      INTEGER SV,SU,SL,CL,P
C
      COMMON /WORK/ SV(5000),IB(250),INB(250),NSV,NIT,NTR
C
      COMMON /DATA/IE,H01,H02,H03,H04,H05,H06,H07,H08,H09,H10,H11,H12,
     1             H21,H22,H23,H24,H31,H32,H41,H42,H43,H44,H51,H52,H53,
     2             H54,H55,H61,H62,H63,H64,H65,H66,H67,H68,H69,H70,H71
     3            ,H72
C
      INTEGER H01,H02,H03,H04,H05,H06,H07,H08,H09,H10,H11,H12,
```

```
     *          H21,H22,H23,H24,H31,H32,H41,H42,H43,H44,H51,H52,
     *          H53,H54,H55,H61,H67,H63,H64,H65,H66,H67,H68,H69,
     *          H70,H71,H72
C
      INTEGER  A(21),PTABLE(300),STABLE(60),CODE(16),ALINE(50)
C
C  LANGUAGE OPTIONS
C
   50 WRITE (6,51)
   51 FORMAT(' ',/)
  100 WRITE(6,101) A
  101 FORMAT(' ',A1,19A4,A3)
      IF (A(2).NE.H33) GO TO 2000
      IF (A(3).NE.H48) GO TO 2500
      IF (A(5) .EQ. H49) GO TO 2750
      MEMORY = A(5)
C
C  OTHER LANGUAGES MAY BE LOADED BY INSERTING SELECTION LOGIC HERE
C
      CALL LODASM(PTABLE)
C
C  LISTING OPTIONS
C
  200 LISTIT=1
      READ(5,201) A
  201 FORMAT(A1,19A4,A3)
      WRITE(6,101) A
      IF (A(2).NE.H45) GO TO 3000
      IF (A(2).NE.H46) GO TO 300
      LISTIT=0
C
C  ORIGIN OPTIONS
C
  300 LCOUNT=0
      READ(5,201) A
      WRITE(6,101) A
      IF (A(2).EQ.H47) GO TO 400
      IF (A(2).NE.H34) GO TO 4000
      IERCOD = 1
      CALL VALREQ (A(3),NUMVAL,IERCOD)
      IF (IERCOD .NE. 0) GO TO 4500
      LCOUNT = NUMVAL
C
C  ASSEMBLY SUBROUTINES
C
  400 WRITE(6,401)
  401 FORMAT(' ',/,25X,'ASSEMBLY BEGINS HERE',/)
      IERCNT=0
      CALL PASONE (LCOUNT,PTABLE,IERCNT)
      END FILE 2
      REWIND 2
C
C  PRINT SYMBOL TABLE  IF DESIRED
C
  500 IF (LISTIT .NE. 1) GO TO 600
      CALL PRINST (STABLE,IPOINT,IER83)
      IF (IPOINT .EQ. 0) GO TO 600
      WRITE (6,501)
  501 FORMAT(' ',//,15X,' SYMBOL TABLE ',/)
      LIMIT = IPOINT / 4
```

```
      DO 550  I = 1,LIMIT,3
      I1 = I + 1
      I2 = I + 2
      WRITE (6,502) STABLE(I), STABLE(I1), STABLE(I2)
  502 FORMAT(' ',10X,'SYMBOL ',A4,A4,' VALUE ',Z4)
  550 CONTINUE
C
CC  WRITE PASS ONE RESULTS IF PASS NOT ERROR FREE
C
  600 IF (IERCNT .EQ. 0) GO TO 700
      WRITE (6,601)
  601 FORMAT(' ',//,25X,'PASS ONE',//)
  650 READ(2,660,END=680)  A
  660 FORMAT(A1,19A4,A3)
      WRITE (6,670)  A
  670 FORMAT(' ',A1,19A4,A3)
      GO TO 650
  680 WRITE (6,690)  IERCNT
  690 FORMAT(' ',5X,'IERCNT = ',I4)
      GO TO 5000
C
CC    PASS TWO PROCESSING
C
  700 WRITE (6,701)
  701 FORMAT(' ',//,25X,'PASS TWO')
      WRITE (6,702)
  702 FORMAT(' ',//,1X,'LC',5X,'CODE',I40,'PROGRAM STATEMENTS',/)
      CALL PASTWO (LCOUNT,PTABLE,STABLE,IERCNT,LISTIT)
      END FILE 3
      IF (IERCNT .EQ. 0) GO TO 800
      WRITE (6,690)  IERCNT
      GO TO 5000
C
CCC   PREPARATION OF CARD IMAGES
CCCC
CCC   INITIALIZE FOR PREPARING CARD IMAGES FOR LOAD PROCESS
CC
  800 REWIND 2
      REWIND 3
  900 CALL IMAGER (MEMORY)
      END FILE 2
      REWIND 2
 1000 WRITE (6,1001)
 1001 FORMAT(' ',//)
C
CC.  SEND RESULTS TO BE READ INTO MEMORY
C
      CALL LODMEM (JN,J1,J2,65000)
      WRITE (6,1001)
      GO TO 6000
C
CC   ERROR MESSAGE ROUTINES
C
 2000 WRITE (6,2001)
 2001 FORMAT(' ','***** MISSING ASM CARD - TERMINATING ASSEMBLY *****')
      GO TO 5000
C
 2500 WRITE (6,2501)
 2501 FORMAT(' ','***** MISSING LANGUAGE ASSIGNMENT - TERMINATING ASSEMB
     CLY *****')
```

```
      GO TO 5000
C
 2750 WRITE (6,2751)
 2751 FORMAT(' ','*****  MISSING MEMORY NAME - TERMINATING ASSEMBLY ****
     C*')
      GO TO 5000
C
 3000 WRITE (6,3001)
 3001 FORMAT(' ','***** LISTING CARD INCORRECT - DEFAULT=LIST *****')
      GO TO 300
C
 4000 WRITE (6,4001)
 4001 FORMAT(' ','***** ORIGIN NOT SPECIFIED - DEFAULT=0 *****')
      GO TO 400
C
 4500 WRITE (6,4501)
 4501 FORMAT(' ','*****  INVALID ORIGIN SPECIFIED - DEFAULT = 0 *****')
      GO TO 400
C
 5000 RETURN 1
C
 6000 WRITE (6,6001)
 6001 FORMAT(' ',10X,'***** END OF ASSEMBLER ROUTINE   *****',/)
      RETURN
10000 STOP
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCC      FORTRAN SUBROUTINE,    TITLE : PASONE       CCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      SUBROUTINE PASONE (LCOUNT,PTABLE,IERCNT)
C
      INTEGER H33,H34,H45,H46,H47,H48,H49,H73
      DATA H33,   H34,   H45,   H46,   H47,   H48,   H49,   H73
     C     /3HASM, 3HORG, 4HLIST,4HNONE,4HNORG,4H8080,4H    ,1H:/
C
      COMMON /DATA/IE,H01,H02,H03,H04,H05,H06,H07,H08,H09,H10,H11,H12,
     1            H21,H22,H23,H24,H31,H32,H41,H42,H43,H44,H51,H52,H53,
     2            H54,H55,H61,H62,H63,H64,H65,H66,H67,H68,H69,H70,H71
     3           ,H72
C
      INTEGER  H01,H02,H03,H04,H05,H06,H07,H08,H09,H10,H11,H12,
     *         H21,H22,H23,H24,H31,H32,H41,H42,H43,H44,H51,H52,
     *         H53,H54,H55,H61,H62,H63,H64,H65,H66,H67,H68,H69,
     *         H70,H71,H72
C
      INTEGER  PASS,A(21),PTABLE(300)
C
C
      PASS=1
      LC=LCOUNT
C
C  READ CARD AND WRITE COPY ON DISK FOR PASS TWO
C
   50 READ(5,51) A
   51 FORMAT(A1,1944,A3)
      WRITE(2,51) A
C
C  PASS COMMENT CARDS BEGINNING WITH * AND FLAG OTHER NONBLANK FIRST
C  COLUMNS AS ERRORS
```

```
C
      IF (A(1).EQ.H03) GO TO 50
      IF (A(1).NE.H01) GO TO 400
C
C  LABEL PROCESSING
C
  100 IF (A(2).EQ.H49) GO TO 200
      IF (A(2).EQ.H31) GO TO 900
      CALL LABLIN (A(2),LC,IERR1)
      IF (IERR1.EQ.1) GO TO 500
      IF (IERR1.EQ.2) GO TO 600
C
C  OPCODE PROCESSING
C
  200 IF (A(4).EQ.H49) GO TO 700
      CALL PCODE (PASS,A(4),PTABLE,LINCR,IERR2)
      IF (IERR2 .EQ. 1 .OR. IERR2 .EQ. 2) GO TO 800
      IF (IERR2.GE.3) GO TO 300
      LC=LC+LINCR
      GO TO 50
C
C  PSUEDO-OP PROCESSING
C
  300 CALL POPSUB(LINCR,PASS,IERR2,A,LC,IDUMMY,&900)
      IF (IERR2 .NE. 0) IERCNT = IERCNT + 1
      GO TO 50
C
C  ERROR MESSAGE ROUTINES
C
  400 WRITE(2,401)
  401 FORMAT('***** COLUMN 1 NOT BLANK OR COMMENT - CARD SKIPPED ****')
      IERCNT=IERCNT+1
      GO TO 50
C
  500 WRITE (2,501) A(2),A(3)
  501 FORMAT('*****  LABEL ',A4,A4,' NOT VALID - CARD SKIPPED *****')
      IERCNT = IERCNT + 1
      GO TO 50
C
  600 WRITE (2,601)
  601 FORMAT('***** LABEL TOO LONG - TRUNCATED TO SIX PLACES *****')
      IERCNT = IERCNT + 1
      GO TO 200
C
  700 IF ( A(6) .EQ. H49) GO TO 50
      WRITE (2,701)
  701 FORMAT('***** OPERANDS APPEAR WITHOUT AN OPCODE - CARD SKIPPED ***
     C**')
      IERCNT = IERCTN + 1
      GO TO 50
C
  800 WRITE (2,801) A(4)
  801 FORMAT('***** OPCODE ',A4,' INVALID OR NOT FOUND - CARD TERMINATED
     C *****')
      IERCNT = IERCNT + 1
      GO TO 50
C
  900 RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

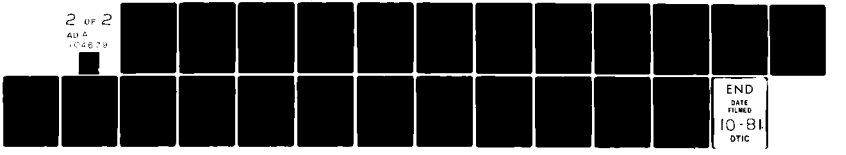
```
CCCCCCCCCC      FORTRAN SUBROUTINE,   TITLE : PASTWO    CCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
      SUBROUTINE PASTWO (LCOUNT,PTABLE,STABLE,IERCNT,LISTIT)
C
C
      INTEGER H03
      DATA  H03/1H*/
C
      INTEGER PASS,A(21),PTABLE(300),STABLE(60),CODE(16),OPER1,OPER2
C
C     INITIALIZATION
C
      PASS = 2
      LC = LCOUNT
      DO 10   I =1,16
      CODE(I) = 0
   10 CONTINUE
C
C   READ CARD IMAGE FROM DISK
C
    50 READ (2,51)  A
    51 FORMAT(A1,19A4,A3)
       IF ( A(1) .NE. H03) GO TO 100
       NBYTES = 0
       GO TO 700
C
C   OPCODE PROCESSING
C
  100 CALL PCODE (PASS,A(4),PTABLE,LINCR,IERR2,MOD,OPER1,OPER2,NUMCOD)
       IF (IERR2 .EQ. 1 .OR. IERR2 .EQ. 2) GO TO 1000
       IF (IERR2 .GE. 3) GO TO 400
       NBYTES = LINCR
  200 CODE(1) = NUMCOD
       IF (OPER1 .EQ. 0) GO TO 600
C
C     OPERAND PROCESSING
C
  300 CALL OPERAN (MOD,A(6),OPER1,IERCOD,OPER2,NVAL1,NVAL2,NVAL3)
       IF (IERCOD .EQ. 1) GO TO 2000
       IF (IERCOD .EQ. 2) GO TO 3000
       IF(MOD .EQ. 0) GO TO 350
       CODE(1) = CODE(1) + NVAL1
       IF (MOD .NE. 5) GO TO 325
       CODE(1) = CODE(1) + NVAL2
       GO TO 600
  325  CODE(2) = NVAL2
       IF (OPER2 .NE. 5) GO TO 600
       CODE(3) = NVAL3
       GO TO 600
  350  CODE(2) = NVAL1
       IF (OPER1 .NE. 5) GO TO 600
       CODE(3) = NVAL2
       GO TO 600
C
C    PSUEDO-OP PROCESSING
C
  400  CALL POPSUB (LINCR,PASS,IERR2,A,LC,CODE,&7000)
       IF (IERR2 .EQ. 1) GO TO 4000
```

```
      IF (IERR2 .EQ. 2) GO TO 5000
      IF (IERR2 .EQ. 3) GO TO 6000
      NBYTES = LINCR
C
C   WRITE LC AND CODE INFOR TO TAPE FOR LOAD TO MEMORY
C
  600 IF (NBYTES .EQ. 0) GO TO 700
      WRITE (3,610)  NBYTES,LC,CODE
  610 FORMAT(Z2,Z4,16Z2)
C
C   IF LIST IS DESIRED, PRINT OUT LC, CODE, AND LINE
C
  700 IF (LISTIT .EQ. 0) GO TO 50
      IF (NBYTES .NE. 0) GO TO 800
C
C   PRINT LINE ONLY FOR COMMENTS AND ZERO LENGTH PSUEDO-OPS
C
  705 WRITE (6,710)  A
  710 FORMAT(' ',T25,A1,19A4,A3)
      GO TO 50
  800 NBYTES = 0
      IF (NBYTES .LE. 3) GO TO 810
      NBYTES = NBYTES
      NBYTES = 3
C
C   PRINT LINE AND CODE FOR NORMAL OP CODES
C
  810 WRITE (6,811) LC,(CODE(I),I=1,NBYTES)
  811 FORMAT(' ',Z4,3(2X,Z2))
      WRITE (6,812)  A
  812 FORMAT('+',T25,A1,19A4,A3)
      IF (NBYTES .EQ. 0) GO TO 900
C
C   PRINT CONTINUED LINES FOR BYTE AND WORD INFORMATION
C
      WRITE (6,813)  (CODE(I), I=4,NBYTES)
  813 FORMAT(' ',T9,13(Z2,2X))
  900 LC = LC + LINCR
      GO TO 50
C
C  ERROR MESSAGES
C
 1000 WRITE (6,710)  A
      WRITE (6,1001)  A(4)
 1001 FORMAT(' ','***** OPCODE ',A4,' INVALID OR NOT FOUND - CARD TERMIN
     CATED *****')
      IERCNT = IERCNT + 1
      LC = LC + LINCR
      GO TO 50
C
 2000 WRITE (6,710)  A
      WRITE (6,2001)
 2001 FORMAT(' ','*****  INVALID OPERAND ENCOUNTERED - CARD TERMINATED *
     C****')
      IERCNT = IERCNT + 1
      LC = LC + LINCR
      GO TO 50
C
 3000 WRITE (6,710)  A
      WRITE (6,3001)
```

```
 3001 FORMAT(' ','*****  OVERLONG OPERAND ENCOUNTERED - CARD TERMINATED
     (*****')
      IERCNT = IERCNT + 1
      LC = LC + LINCR
      GO TO 50
C
 4000 WRITE (6,710)  A
      WRITE (6,4001)
 4001 FORMAT(' ','***** INVALID OPERAND - NOT PROCESSED *****')
      IERCNT = IERCNT + 1
      LC = LC + LINCR
      GO TO 50
C
 5000 WRITE (6,710)  A
      WRITE (6,5001)
 5001 FORMAT(' ','***** OVERLONG OPERAND - NOT PROCESSED *****')
      IERCNT = IERCNT + 1
      LC = LC + LINCR
      GO TO 50
C
 6000 WRITE (6,710)  A
      WRITE (6,6001)
 6001 FORMAT(' ','***** LANGUAGE OR IERR2 ERROR IN POPSUB *****')
      IERCNT = IERCNT + 1
      LC = LC + LINCR
      GO TO 50
C
C  ROUTINE EXIT
C
 7000 IF (LISTIT .EQ. 0) GO TO 7010
      WRITE (6,710)  A
 7010 RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCC      FORTRAN SUBROUTINE,   TITLE : POPSUB      CCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      SUBROUTINE POPSUB (LINCR,PASS,IERR2,A,LC,CODE,*)
C
C
      INTEGER A(21),PASS,CODE(16)
C
C  USE LINCR TO IDENTIFY LANGUAGE
C
    1 IF (LINCR.NE.1) GO TO 1700
      IF (PASS.EQ.2) GO TO 900
C
C  DB PROCESSING
C
  100 IF (IERR2.NE.3) GO TO 200
      ITYPE = 1
      CALL STRING (ITYPE,PASS,A(6),NUMOPS,IERCOD)
      IF (IERCOD.EQ.1) GO TO 1500
      IF (IERCOD.EQ.2) GO TO 1600
      IERR2 = 0
      LC = LC+ NUMOPS
      RETURN
C
C  DS PROCESSING
C
```

```
  200 IF (IERR2.NE.4) GO TO 300
      IERCOD = 1
      CALL VALREQ (A(6),NUMVAL,IERCOD)
      IF (IERCOD.EQ.1) GO TO 1500
      IF (IERCOD.EQ.2) GO TO 1600
      IERR2 = 0
      LC = LC + NUMVAL
      RETURN
C
C   DW PROCESSING
C
  300 IF (IERR2.NE.5) GO TO 400
      ITYPE = 2
      CALL STRING (ITYPE,PASS,A(6),NUMOPS,IERCOD)
      IF (IERCOD.EQ.1) GO TO 1500
      IF (IERCOD.EQ.2) GO TO 1600
      IERR2 = 0
      LC = LC + (NUMOPS * 2)
      RETURN
C
C   END PROCESSING, ENDING PASS ONE
C
  400 IF (IERR2.NE.6) GO TO 500
      RETURN1
C
C   EQU PROCESSING
C
  500 IF (IERR2.NE.7) GO TO 600
      IERCOD = 1
      CALL VALREQ (A(6),NUMVAL,IERCOD)
      IF (IERCOD.EQ.1) GO TO 1500
      IF (IERCOD.EQ.2) GO TO 1600
      CALL LABFIX (A(2),NUMVAL,IERCOD)
      IF (IERCOD.EQ.1) GO TO 1500
      IF (IERCOD.EQ.2) GO TO 1600
      IERR2 = 0
      RETURN
C
C   ORG PROCESSING
C
  600 IF (IERR2.NE.8) GO TO 700
      IERCOD = 1
      CALL VALREQ (A(6),NUMVAL,IERCOD)
      IF (IERCOD.EQ.1) GO TO 1500
      IF (IERCOD.EQ.2) GO TO 1600
      IERR2 = 0
      LC = NUMVAL
      RETURN
C
C   RST PROCESSING
C
  700 IF (IERR2 .NE. 9) GO TO 1700
      IERCOD = 1
      CALL VALREQ (A(6),NUMVAL,IERCOD)
      IF (IERCOD.EQ.1) GO TO 1500
      IF (IERCOD.EQ.2) GO TO 1600
      IERR2 = 0
      LC = LC + 1
      RETURN
C
```

AD-A104 679 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH F/G 9/2
FIDELITY OPTIMIZATION OF MICROPROCESSOR SYSTEM SIMULATIONS.(U)
MAR 81 E T LANDRUM
UNCLASSIFIED AFIT-CI-81-3T NL



```
C
C    PASS TWO ROUTINES
C
C
C
C    DB PROCESSING
C
  800 IF (IERR2 .NE. 3) GO TO 900
      ITYPE = 1
      CALL STRING (ITYPE,PASS,A(6),NUMOPS,IERCOD,CODE)
      IERR2 = IERCOD
      IF (IERR2 .GT. 0) RETURN
      LINCR = NUMOPS
      RETURN
C
C    DS PROCESSING
C
  900 IF (IERR2.NE.4) GO TO 1000
      IERCOD = 1
      CALL VALRED (A(6),NUMVAL,IERCOD)
      IERR2 = IERCOD
      IF (IERR2 .GT. 0) RETURN
      LINCR = NUMVAL
      RETURN
C
C    DW PROCESSING
C
 1000 IF (IERR2.NE.5) GO TO 1100
      ITYPE = 2
      CALL STRING (ITYPE,PASS,A(6),NUMOPS,IERCOD,CODE)
      IERR2 = IERCOD
      IF (IERR2 .GT. 0) RETURN
      LINCR = NUMOPS * 2
      RETURN
C
C    END PROCESSING, END OF PASS TWO
C
 1100 IF (IERR2.NE.6) GO TO 1200
      LINCR = 0
      RETURN1
C
C    EQU PROCESSING
C
 1200 IF (IERR2.NE.7) GO TO 1300
      LINCR = 0
      IERR2 = 0
      RETURN
C
C    ORG PROCESSING
C
 1300 IF (IERR2.NE.8) GO TO 1400
      IERCOD = 1
      CALL VALRED (A(6),NUMVAL,IERCOD)
      IERR2 = IERCOD
      IF (IERR2 .GT. 0) RETURN
      LINCR = 0
      LC = NUMVAL
      RETURN
C
C    RST PROCESSING
```

```
C
 1400 IF (IERR2 .NE. 9) GO TO 1800
      IERCOD = 1
      CALL VALRED (A(6),NUMVAL,IERCOD)
      IERR2 = IERCOD
      IF (NUMVAL .GT. 7 .OR. NUMVAL .LT. 1) IERR2 = 2
      IF (IERR2 .GT. 0) RETURN
      NUMVAL = NUMVAL * 8
      CODE(1) = 199 + NUMVAL
      LINCR = 1
      RETURN
C
C  ERROR MESSAGES
C
 1500 WRITE(2,1501)
 1501 FORMAT('*****  INVALID OPERAND - NOT PROCESSED  *****')
      IERR2 = 1
      RETURN
C
 1600 WRITE(2,1601)
 1601 FORMAT('*****  OPERAND LENGTH EXCEEDED - NOT PROCESSED  *****')
      IERR2 = 2
      RETURN
C
 1700 WRITE(2,1701)
 1701 FORMAT('***** LANGUAGE OR IERR2 ERROR IN POPSUB *****')
      IERR2 = 2
      RETURN
C
 1800 IERR2 = 3
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCC      FORTRAN SUBROUTINE,   TITLE : IMAGER          CCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE IMAGER (MEMORY)
C
C  SUBROUTINE TO PLACE ASSEMBLED CODE IN FORMAT CARD IMAGE,
C  TO BE LOADED INTO THE COL LOAD MODULE
C
      COMMON /DATA/IE,H01,H02,H03,H04,H05,H06,H07,H08,H09,H10,H11,H12,
     1              H21,H22,H23,H24,H31,H32,H41,H42,H43,H44,H51,H52,H53,
     2              H54,H55,H61,H62,H63,H64,H65,H66,H67,H68,H69,H70,H71
     3             ,H72
C
      INTEGER H01,H02,H03,H04,H05,H06,H07,H08,H09,H10,H11,H12,
     *        H21,H22,H23,H24,H31,H32,H41,H42,H43,H44,H51,H52,
     *        H53,H54,H55,H61,H62,H63,H64,H65,H66,H67,H68,H69,
     *        H70,H71,H72
C
      INTEGER  CODE(16),ALINE(50)
      DATA  ICOLON/':'/
C
C  SET INITIAL CONSTANTS
C
      ISPLIT = 0
      IFIRST = 1
      NEWLIN = 1
      K = 0
C
```

```
C  READ LINE OF CODE
C
   10 READ (3,20,END=600)  NBYTES,LC,CODE
   20 FORMAT(Z2,Z4,16Z2)
      L = 0
      IF (K .EQ. 14)  NEWLIN = 1
      IF (NEWLIN .EQ. 0) GO TO 300
C
C  WRITE FINISHED LINE AND CREATE NEW LINE HEADING
C
  100 IF (IFIRST .EQ. 1) GO TO 150
      MAX = M - 1
      WRITE (2,110)  (ALINE(I), I=1,MAX)
  110 FORMAT(1A1,A4,2A1,Z4,4A1,Z2,13(2A1,Z2))
  150 IFIRST = 0
      IF (ISPLIT .EQ. 1) LC = LC1
      ISPLIT = 0
      LC1 = LC
      K = 0
      M = 9
  200 ALINE(1) = H01
      ALINE(2) = MEMORY
      ALINE(3) = H09
      ALINE(4) = ICOLON
      ALINE(5) = LC1
      ALINE(6) = H12
      ALINE(7) = H10
      ALINE(8) = H11
      ICOMMA = 0
      NEWLIN = 0
C
C  TEST FOR NON-CONTIGUOUS LC
C
  300 IF (LC1 .NE. LC) GO TO 100
C
C  TEST FOR FIRST NUMBER, WHICH NEEDS NO COMMA
C
  400 IF (ICOMMA .EQ. 0) GO TO 500
C
C  ADD COMMA TO LINE
C
      ALINE(M) = H08
      M = M + 1
C
C  ADD ONE BYTE OF CODE
C
  500 ICOMMA = 1
      ALINE(M) = ICOLON
      M = M + 1
      L = L + 1
      ALINE(M) = CODE(L)
      LC1 = LC1 + 1
      K = K + 1
      M = M + 1
C
C  TEST FOR END OF CODE STRING AND FOR FILLED LINE
C
      IF (L .EQ. NBYTES) GO TO 10
      IF (K .NE. 14) GO TO 400
      ISPLIT = 1
```

```
      GO TO 100
  600 MAX = M - 1
      WRITE (2,110)  (ALINE(I), I=1,MAX)
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCC      FORTRAN SUBROUTINE,   TITLE : LODMEM                          CCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE LODMEM (JN,J1,J2,*)
      DIMENSION JN(9)
      DATA IEQS,ICOM / '=',',' /
      COMMON /TABLES/ SV(7,15),SL(3,15),LB(4,250),CL(4),
     *          NSV,NSL,NLB,NCL,NP,NST,NSY,NPSAVE,MAXP,P(1)
      INTEGER SV,SU,SL,CL,P,A(21)
      COMMON /WORK/ SV(5000),IB(250),INB(250),NSV,NIT,NTR
      COMMON /DATA/IE,H01,H02,H03,H04,H05,H06,H07,H08,H09,H10,H11,H12,
     1             H21,H22,H23,H24,H31,H32,H41,H42,H43,H44,H51,H52,H53,
     2             H54,H55,H61,H62,H63,H64,H65,H66,H67,H68,H69,H70,H71
     3            ,H72
C
      INTEGER H01,H02,H03,H04,H05,H06,H07,H08,H09,H10,H11,H12,
     *        H21,H22,H23,H24,H31,H32,H41,H42,H43,H44,H51,H52,
     *        H53,H54,H55,H61,H67,H63,H64,H65,H66,H67,H68,H69,
     *        H70,H71,H72
C
   10 READ (2,11,END=6000)  A
   11 FORMAT(A1,19A4,A3)
   20 WRITE (6,21) A
   21 FORMAT(' ',A1,19A4,A3)
      IF (A(1) .NE. H01) RETURN
      CALL SEPT(A(2),IA,IB,INB,N)
      I=1
   40 IF (I.GE.N) GO TO 10
      IF (IB(I) .EQ. H08) I = I + 1
      IF (INB(I).NE.1) GO TO 4000
      JN(2)=IA(I)
      MS=0
      CALL FNAME(JN,MS,&4100)
      IF (JN(1).NE.1) GO TO 50
      GO TO 4000
C
C                     MEMORY
C
   50 IF (JN(1) .NE. 3 .OR. IB(I+1) .NE. H09) GO TO 4100
      IF (INB(I+2).EQ.2) GO TO 80
      IF (J1 .LT. 0) GO TO 4000
   60 IF (IB(I+2) .NE. H12) GO TO 4000
      J2=0
      IF (INB(I+3).NE.2) GO TO 70
      J2=IB(I+3)
      I=I+1
   70 IF (IB(I+3) .NE. H10) GO TO 4000
      I=I+3
      GO TO 90
   80 J1=IB(I+2)
      I=I+1
      IF (IB(I+2) .NE. H10) GO TO 60
      J2=J1
      I=I+2
   90 IAA=IEQS
```

```
C
  100 IF ((I+2).GT.N) GO TO 10
      IF (IA(I+1) .NE. IAA) GO TO 4000
      IF (INR(I+2).NE.2) GO TO 110
      SV(NSV+1)=-101
      SV(NSV+2)=JN(3)
      SV(NSV+3)=0
      SV(NSV+4)=IA(I+2)
      SV(NSV+5)=JN(3)
      SV(NSV+6)=J1*JN(3)+JN(4)
      NSV=NSV+6
      J1=1+J1
      I=I+2
      IAA=ICOM
      IF (J2.EQ.0) GO TO 100
      IF (J1.LE.J2) GO TO 100
  110 I=I+2
      GO TO 40
 4000 CALL ERROR (IE,1,42,&5000)
 4100 CALL ERROR (IE,2,42,&5000)
 5000 RETURN 1
 6000 RETURN
10000 STOP
      END
```

APPENDIX D

PREPROCESSOR ASSEMBLY LANGUAGE ROUTINES

```
**********************************************************************
**********      ASSEMBLER SUBROUTINE,   TITLE : LODASM      **********
**********      LODASM (PTABLE)                             **********
**********************************************************************
LODASM   CSECT
         $EQUR                              EQUATE REGISTERS
         $OPEN                              IBM OPENING CONVENTIONS
BEGIN    EQU   *
         L     R2,0(R1)                     LOAD ADDR OF ARRAY PTALBE
*
*  MOVE CONSTANTS TO PTABLE IN MAIN ROUTINE
*
         MVC   0(256,R2),PTABLE
         MVC   255(256,R2),PTABLE+255
         MVC   511(256,R2),PTABLE+511
         MVC   767(256,R2),PTABLE+767
         MVC   1023(8,R2),PTABLE+1023
         $CLOSE
         DS    0F                           ALIGN TO FULL WORD BOUNDARY
PTABLE   EQU   *
         DC     CL8'ACI  2040'
         DC    F'206'
         DC     CL8'ADC  1110'
         DC    F'136'
         DC     CL8'ADD  1110'
         DC    F'129'
         DC     CL8'ADI  2040'
         DC    F'198'
         DC     CL8'ANA  1110'
         DC    F'160'
         DC     CL8'ANI  2040'
         DC    F'230'
         DC     CL8'CALL 3050'
         DC    F'205'
         DC     CL8'CC   3050'
         DC    F'220'
         DC     CL8'CM   3050'
         DC    F'252'
         DC     CL8'CMA  1000'
         DC    F'47'
         DC     CL8'CMC  1000'
         DC    F'63'
         DC     CL8'CMP  1110'
         DC    F'184'
         DC     CL8'CNC  3050'
         DC    F'212'
         DC     CL8'CNZ  3050'
         DC    F'196'
         DC     CL8'CP   3050'
         DC    F'244'
         DC     CL8'CPE  3050'
         DC    F'236'
```

```
DC      CL8'CPI  2040'
DC     F'254'
DC      CL8'CPO  3050'
DC     F'228'
DC      CL8'CZ   3050'
DC     F'204'
DC      CL8'DAA  1000'
DC     F'39'
DC      CL8'DAD  1320'
DC     F'9'
DC      CL8'DB   P1   '
DC     F'3'
DC      CL8'DCR  1210'
DC     F'5'
DC      CL8'DCX  1320'
DC     F'11'
DC      CL8'DI   1000'
DC     F'243'
DC      CL8'DS   P1   '
DC     F'4'
DC      CL8'DW   P1   '
DC     F'5'
DC      CL8'EI   1000'
DC     F'251'
DC      CL8'END  P1   '
DC     F'6'
DC      CL8'EQU  P1   '
DC     F'7'
DC      CL8'HLT  1200'
DC     F'118'
DC      CL8'IN   2040'
DC     F'219'
DC      CL8'INR  1210'
DC     F'4'
DC      CL8'INX  1320'
DC     F'3'
DC      CL8'JC   3050'
DC     F'218'
DC      CL8'JM   3050'
DC     F'250'
DC      CL8'JMP  3050'
DC     F'195'
DC      CL8'JNC  3050'
DC     F'210'
DC      CL8'JNZ  3050'
DC     F'194'
DC      CL8'JP   3050'
DC     F'242'
DC      CL8'JPE  3050'
DC     F'234'
DC      CL8'JPO  3050'
DC     F'226'
DC      CL8'JZ   3050'
DC     F'202'
DC      CL8'LDA  3050'
DC     F'58'
DC      CL8'LDAX 1430'
DC     F'10'
DC      CL8'LHLD 3050'
DC     F'42'
```

```
DC      CL8'LXI  3325'
DC      F'1'
DC      CL8'MOV  1511'
DC      F'64'
DC      CL8'MVI  2214'
DC      F'6'
DC      CL8'NOP  1000'
DC      F'0'
DC      CL8'ORA  1110'
DC      F'176'
DC      CL8'ORG  P1  '
DC      F'8'
DC      CL8'ORI  2040'
DC      F'246'
DC      CL8'OUT  2040'
DC      F'211'
DC      CL8'PCHL1000'
DC      F'233'
DC      CL8'POP  1320'
DC      F'193'
DC      CL8'PUSH1320'
DC      F'197'
DC      CL8'RAL  1000'
DC      F'23'
DC      CL8'RAR  1000'
DC      F'31'
DC      CL8'RC   1000'
DC      F'216'
DC      CL8'RET  1000'
DC      F'201'
DC      CL8'RIM  1000'
DC      F'32'
DC      CL8'RLC  1000'
DC      F'7'
DC      CL8'RM   1000'
DC      F'248'
DC      CL8'RNC  1000'
DC      F'208'
DC      CL8'RNZ  1000'
DC      F'192'
DC      CL8'RP   1000'
DC      F'240'
DC      CL8'RPE  1000'
DC      F'232'
DC      CL8'RPO  1000'
DC      F'224'
DC      CL8'RRC  1000'
DC      F'15'
DC      CL8'RST  P1  '
DC      F'9'
DC      CL8'RZ   1000'
DC      F'200'
DC      CL8'SBB  1110'
DC      F'152'
DC      CL8'SBI  1040'
DC      F'222'
DC      CL8'SHLD3050'
DC      F'34'
DC      CL8'SIM  1000'
DC      F'48'
```

```
         DC      CL8'SPHL1000'
         DC      F'249'
         DC      CL8'STA 3050'
         DC      F'50'
         DC      CL8'STAX1430'
         DC      F'2'
         DC      CL8'STC 1000'
         DC      F'55'
         DC      CL8'SUB 1110'
         DC      F'144'
         DC      CL8'SUI 2040'
         DC      F'214'
         DC      CL8'XCHG1000'
         DC      F'235'
         DC      CL8'XRA 1110'
         DC      F'168'
         DC      CL8'XRI 2040'
         DC      F'238'
         DC      CL8'XTHL1000'
         DC      F'227'
END      EQU     *
         END     LOOASM
/*
//STEP6 EXEC ASMFC,PARM='NODECK,LOAD'
//ASM.SYSLIB DD DSN=IE17D.MACLIB,DISP=SHR
//          DD DSN=SYS1.MACLIB,DISP=SHR
//ASM.SYSPRINT DD DUMMY
//ASM.SYSGO DD DSN=&LOADJECT,DISP=(MOD,PASS)
//ASM.SYSIN DD *
*********************************************************************
**********      ASSEMBLER SUBROUTINE,   TITLE : LABLST     **********
*********************************************************************
LABLST   CSECT
         ENTRY LABLIN,PRINST,LABFIX,LABOUT
         $EQUR                          EQUATE REGISTERS
         USING *,R15                    ESTABLISH ADDRESSABILITY
*
*  ROUTINE FOR PLACING LABELS AND VALUES INTO SYMBOL TABLE
*        LABLIN (A(2),LC,(ERR))
*
LABLIN   EQU     *
         STM     R14,R12,12(R13)        SAVE GENERAL REGISTERS
         LA      R12,0                  ZERO ERROR CODE
         L       R2,0(R1)               LOAD ADDR OF LABEL
         MVC     SYMBIN(8),0(R2)        LOAD LABEL TO SYMBIN
*
*  TEST FOR ALPHABETIC FIRST CHARACTER
*
         CLI     SYMBIN,C'A'            FIRST CHARACTER A?
         BL      ERRINV                 INVALID IF LOW
         CLI     SYMBIN,C'Z'            FIRST CHARACTER Z?
         BNH     FILLUP                 PROCEED TO FILLUP IF NOT HIGH
         B       ERRINV                 ELSE CHAR INVALID
*
*  LOOP TO LOAD CHARACTERS INTO SYMBOL
*
FILLUP   MVC     SYMBOL(1),SYMBIN       LOAD FIRST CHAR OF SYMBIN
         MVC     SYMBOL+1(7),BLANKS     FILL REST WITH BLANKS
         LA      R4,0                   ZERO POINTER
         LA      R8,1                   SET INCREMENT
```

```
         LA    R9,5                    SET LIMIT INDEX
         LA    R5,0                    ZERO R5
FILOOP   IC    R5,SYMBIN+1(R4)         INSERT CHAR FROM SYMBIN
         C     R5,=F'122'              CHARACTER :?
         BE    OUT                     END OF LABEL
         BXH   R4,R8,ERLONG            IF OVER 6 CHAR, ERROR LONG
         STC   R5,SYMBOL(R4)           STORE CHAR IN SYMBOL
         B     FILOOP                  GET NEXT CHARACTER
*
*  LOAD LABEL AND VALUE IN STABLE
*
OUT      EQU   *
         L     R6,POINTR               LOAD INDEX TO STABLE
         LA    R7,STABLE               LOAD ADDR OF STABLE
         AR    R7,R6                   FORM TABLE ADDRESS
         MVC   0(8,R7),SYMBOL          MOVE SYMBOL TO STABLE
         LA    R7,8(R7)                INCREMENT POSITION
         L     R3,4(R1)                LOAD ADDR OF LC
         MVC   0(8,R7),0(R3)           LOAD LC INTO STABLE
         LA    R6,12(R6)               ADJUST INDEX
         ST    R6,POINTR               STORE POINTER
         B     LEAVE                   LEAVE ROUTINE
*
*   ROUTINE TO SEND STABLE TO MAIN STORAGE FOR PRINTING
*
         USING *,R15                   ESTABLISH ADDRESSABILITY
PRINST   EQU   *
         STM   R14,R12,12(R13)         SAVE GENERAL REGISTERS
         LA    R12,0                   ZERO ERROR CODE
MOVING   EQU   *
         L     R2,0(R1)                LOAD ADDR OF STABLE  MAIN PROG
         MVC   0(240,R2),STABLE        MOVE ENTIRE TABLE
         L     R2,4(R1)                LOAD ADDR OF IPOINT
         L     R3,POINTR               LOAD POINTR
         ST    R3,0(R2)                PASS POINTR AS IPOINT
         B     LEAVE
*
*  ROUTINE TO PROCESS EQU PSUEDO-OP
*
         USING *,R15                   ESTABLISH ADDRESSABILITY
LABFIX   EQU   *
         STM   R14,R12,12(R13)         SAVE GENERAL REGISTERS
         LA    R12,0                   ZERO ERROR CODE
         LA    R4,STABLE               LOAD ADDR OF STABLE
         L     R6,POINTR               LOAD POINTR TO R6
         S     R6,=F'12'               POINT TO LAST TABLE ENTRY
         AR    R4,R6                   FORM ADDR OF ELEMENT
         L     R2,4(R1)                LOAD ADDR VALUE
         L     R2,0(R2)                LOAD VALUE
         LA    R4,8(R4)                INCREMENT POINTR TO LC SPACE
         ST    R2,0(R4)                LOAD REVISED VALUE
         B     LEAVE                   AND LEAVE ROUTINE
*
*  ROUTINE TO FIND VALUE FROM SYMBOLS USED
*
         USING *,R15                   ESTABLISH ADDRESSABILITY
LABOUT   EQU   *
         STM   R14,R12,12(R13)         SAVE GENERAL REGISTERS
         LA    R12,0                   ZERO ERROR CODE
         L     R2,0(R1)                LOAD ADDR LABNUM
```

```
         LA    R4,STABLE                LOAD ADDR OF STABLE
         L     R5,POINTR                LOAD LIMIT OF STABLE INDEX
         C     R5,=F'0'                 POINTR = 0 ?
         BE    ERRINV                   IF YES, GO TO ERRINV
         AR    R5,R4                    SET LIMIT ADDRESS
SEEK     EQU   *
         CLC   0(6,R2),0(R4)            COMPARE TO STABLE ENTRY
         BE    FIND                     IF EQUAL, GO TO FIND
         LA    R4,12(R4)                ELSE INCREMENT ADDR
         CR    R4,R5                    COMPARE ADDR TO LIMIT
         BE    ERRINV                   IF EQUAL, GO TO INVALID
         B     SEEK                     ELSE BRANCH BACK
FIND     EQU   *
         LA    R4,8(R4)                 INCREMENT TO LC SPACE
         L     R6,0(R4)                 LOAD LC
         L     R2,4(R1)                 LOAD ADDR NUMVAL
         ST    R5,0(R2)                 PASS BACK NUMVAL
         B     LEAVE                    AND LEAVE ROUTINE
*
*  ERROR HANDLING ROUTINES
*
ERRINV   EQU   *
         LA    R12,1                    SET ERROR CODE = 1
         B     LEAVE                    LEAVE ROUTINE
ERLONG   EQU   *
         LA    R12,2                    SET ERROR CODE = 2
*
*  LOAD ERROR CODE AND LEAVE ROUTINE
*
LEAVE    EQU   *
         L     R6,8(R1)                 LOAD ADDR OF ERROR CODE
         ST    R12,0(R6)                PASS BACK ERROR CODE
         LM    R14,R12,12(R13)          RESTORE GENERAL REGISTERS
         BR    R14                      RETURN
SYMBOL   DS    1D
SYMBIN   DS    1D
POINTR   DC    F'0'
BLANKS   DC    CL8' '
STABLE   DC    240CL1' '                SPACE FOR 20 ENTIRES IN STABLE
         LTORG
         END   LABLST
/*
//STEP7 EXEC ASMFC,PARM='NODECK,LOAD'
//ASM.SYSLIB DD DSN=IEI70.MACLIB,DISP=SHR
//         DD DSN=SYS1.MACLIB,DISP=SHR
//ASM.SYSPRINT DD DUMMY
//ASM.SYSGO DD DSN=&&OBJECT,DISP=(MOD,PASS)
//ASM.SYSIN DD *
*********************************************************************
**********      ASSEMBLER SUBROUTINE,   TITLE : PCODE      **********
**********      PCODE (PASS,A(4),TABLE,LINCR,(ERR2)        **********
*********************************************************************
PCODE    CSECT
         $EQUR                          EQUATE REGISTERS
         $OPEN                          IBM OPENING CONVENTIONS
BEGIN    EQU   *
         LA    R10,0                    ZERO OFFSET
         LA    R12,0                    ZERO ERROR CODE
         ST    R12,LEN                  ZERO LINCR
         L     R2,4(R1)                 R2 IS PASS ADDR
```

```
         L      R2,0(R2)                 R2 IS PASS
         L      R3,4(R1)                 R3 IS SEARCH PCODE ADDR
         L      R6,0(R3)                 R6 IS SEARCH PCODE
*
*  ERROR TEST AND OFFSET DETERMINATION
TEST1    CLI    0(R3),C'Z'               PCODE GREATER THAN Z?
         BNH    TEST2                    IF NOT, GO TO TEST2
ERROR1   LA     R12,1                    IF YES, ERROR CODE = 1
         B      LEAVE                    LEAVE ROUTINE
TEST2    CLI    0(R3),C'Q'               PCODE GREATER THAN Q?
         BNH    TEST3                    IF NOT, GO TO TEST3
         LA     R10,684                  IF YES,SEARCH BEGINS AT Q
         B      SEARCH                   GO TO SEARCH
TEST3    CLI    0(R3),C'G'               PCODE GREATER THAN G?
         BNH    TEST4                    IF NOT, GO TO TEST4
         LA     R10,360                  IF YES, SEARCH BEGINS AT G
         B      SEARCH                   GO TO SEARCH
TEST4    CLI    0(R3),C'A'               PCODE GREATER THAN A?
         BL     ERROR1                   IF NOT, ERROR CODE =1
*
*  SEARCH LOOP
*
SEARCH   LA     R4,30                    LIMIT OF 30 PROBES
         L      R5,8(R1)                 LOAD ADDR OF PTABLE
SLOOP    C      R6,0(R10,R5)             COMPARE ENTRY TO PCODE TABLE
         BE     FOUND                    IF EQUAL, GO TO FOUND
         BL     NOTFND                   IF SMALLER, BRANCH OUT
         LA     R10,12(R10)              IF NOT, INCREMENT POINTER
         BCT    R4,SLOOP                 BRANCH ON COUNT TO SLOOP
NOTFND   LA     R12,2                    IF NOT FOUND, SET ERROR CODE
         B      LEAVE                    AND LEAVE ROUTINE
FOUND    EQU    *
         AR     R5,R10                   SUM BASE AND INDEX
         LA     R5,4(R5)                 MOVE TO LENGTH BYTE
         CLI    0(R5),C'P'               TEST FOR PSUEDO-OP
         BNE    ONEPAS                   IF NOT PSUEDO-OP, GO TO ONEPAS
         L      R12,4(R5)                SET ERROR CODE FOR PSUEDO-OP
         LA     R5,1(R5)                 MOVE TO MOD BYTE
         MVC    LEN+3(1),0(R5)           MOVE LENGTH TO LEN
         NI     LEN+3,X'0F'              CONVERT TO BINARY
         L      R6,LEN                   LOAD LEN TO R6
         L      R4,12(R1)                LOAD ADDR OF LINCR
         ST     R6,0(R4)                 PASS BACK AS LINCR
         B      LEAVE                    AND LEAVE ROUTINE
*
*  PASS ONE PROCESSING
*
ONEPAS   EQU    *
         MVC    LEN+3(1),0(R5)           MOVE LENGTH TO LEN
         NI     LEN+3,X'0F'              CONVERT TO BINARY
         L      R6,LEN                   LOAD LEN TO R6
         L      R4,12(R1)                LOAD ADDR OF LINCR
         ST     R6,0(R4)                 PASS BACK AS LINCR
         C      R2,=F'2'                 IF PASS=2
         BE     TWOPAS                   GO TO PASS TWO HANDLING
         B      LEAVE                    ELSE LEAVE ROUTINE
*
*  PASS TWO PROCESSING
*  PASS BACK MOD, OPER1, OPER2, AND NUMCOD
*
```

```
TWOPAS   EQU   *
MOD      EQU   *
         LA    R5,1(R5)                  ADVANCE POINTER TO MOD
         MVC   LEN+3(1),0(R5)            MOVE MOD TO LEN
         NI    LEN+3,X'0F'               CONVERT TO BINARY
         L     R6,LEN                    LOAD MOD
         L     R4,20(R1)                 LOAD ADDR OF MOD
         ST    R6,0(R4)                  PASS BACK MOD
OPER1    EQU   *
         LA    R5,1(R5)                  ADVANCE POINTER TO OPER1
         MVC   LEN+3(1),0(R5)            LOAD OPER1 INTO LEN
         NI    LEN+3,X'0F'               CONVERT TO BINARY
         L     R6,LEN                    LOAD OPER1
         L     R4,24(R1)                 LOAD ADDR OPER1
         ST    R6,0(R4)                  PASS BACK OPER1
OPER2    EQU   *
         LA    R5,1(R5)                  ADVANCE POINTER TO OPER2
         MVC   LEN+3(1),0(R5)            MOVE OPER2 TO LEN
         NI    LEN+3,X'0F'               CONVERT TO BINARY
         L     R6,LEN                    LOAD OPER2
         L     R4,28(R1)                 LOAD ADDR OPER2
         ST    R6,0(R4)                  PASS BACK OPER2
NUMCOD   EQU   *
         LA    R5,1(R5)                  ADVANCE POINTER TO NUMCOD
         L     R6,0(R5)                  LOAD NUMCOD
         L     R4,32(R1)                 LOAD ADDR NUMCOD
         ST    R6,0(R4)                  PASS BACK NUMCOD
LEAVE    L     R6,16(R1)                 LOAD RETURN ADDR
         ST    R12,0(R6)                 PASS BACK ERROR CODE
END      EQU   *
         $CLOSE                          IBM CLOSING CONVENTIONS
LEN      DS    1F
         END   PCODE
/*
//STEP8 EXEC ASMFC,PARM='NODECK,LOAD'
//ASM.SYSLIB DD DSN=IE170.MACLIB,DISP=SHR
//           DD DSN=SYS1.MACLIB,DISP=SHR
//ASM.SYSPRINT DD DUMMY
//ASM.SYSGO  DD DSN=&&OBJECT,DISP=(MOD,PASS)
//ASM.SYSIN  DD *
*********************************************************************
**********      ASSEMBLER SUBROUTINE,  TITLE : STRING       **********
**********      CALL STRING (ITYPE,PASS,A(6),NUMCPS,IERCOD,CODE)  *****
*********************************************************************
STRING   CSECT
         $EQUR                           EQUATE REGISTERS
         $OPEN                           IBM OPENING CONVENTIONS
BEGIN    EQU   *
         LA    R12,0                     ZERO ERROR CODE
         ST    R12,ERCODE                ZERO ERROR CODE FOR VALRED
*    SET LIMIT BY TYPE
         L     R2,0(R1)                  LOAD ITYPE ADDR
         L     R2,0(R2)                  LOAD ITYPE
         LR    R9,R2                     STORE ITYPE IN R9
         LA    R3,16                     LOAD LIMIT TO R3
         ST    R3,LIMIT                  STORE LIMIT
         C     R2,=F'1'                  ITYPE = 1 ?
         BE    TESTIT                    IF YES, GO TO TESTIT
         LA    R3,8                      ELSE USE LIMIT FOR WORDS, TYPE 2
         ST    R3,LIMIT                  STORE AS LIMIT
```

```
*   INITIALIZE BY PASS CODE
TESTIT   EQU   *
         L     R2,8(R1)                  LOAD ADDR OF POSITION
         L     R3,4(R1)                  LOAD ADDR PASS
         L     R3,0(R3)                  LOAD PASS
         C     R3,=F'1'                  PASS = 1 ?
         BE    TSTINV                    IF SO, GO TO VALIDITY TEST
         L     R10,20(R1)                ELSE LOAD ADDR CODE
         LA    R11,0                     AND LOAD INDEX TO CODE
         ST    R2,ADDROP                 STORE POSITION IN ADDROP
*   TEST FIRST CHARACTER
TSTINV   EQU   *
         CLI   0(R2),C' '                FIRST CHAR BLANK ?
         BE    ERRINV                    IF YES, OPERAND INVALID
         CLI   0(R2),C''''               FIRST CHAR ' ?
         BE    QUOTES                    GO TO STRING PROCESSING
         CLI   0(R2),C','                FIRST CHAR , ?
         BE    ERRINV                    IF YES, OPERAND INVALID
         LA    R4,1                      SET COUNT TO 1
*   MAIN TEST LOOP
COMCNT   EQU   *
         CLI   0(R2),C' '                CHARACTER BLANK ?
         BE    DONE                      IF YES, GO TO DONE
         CLI   0(R2),C','                CHARACTER , ?
         BE    OPONE                     IF YES, ADD ONE OPERAND
         B     INCR                      COUNT NEXT OPERAND
*   INCREMENT AND LOAD ON PASS 2
OPONE    EQU   *
         LA    R4,1(R4)                  INCREMENT OPERANDS
         C     R4,LIMIT                  LONGER THAN LIMIT
         BH    ERLONG                    IF YES, GO TO ERROP LONG
         C     R3,=F'1'                  PASS = 1 ?
         BE    INCR                      IF YES, GO TO INCR
SUBOUT   EQU   *
         LR    R7,R1                     ELSE SAVE R1
         CALL  VALRED,(ADDROP,NUMVAL,ERCODE),VL
         LR    R1,R7                     RESTORE R1
         L     R6,ERCODE                 LOAD ERROR CODE
         C     R6,=F'0'                  ERCODE = 0 ?
         BE    PASBAK                    IF YES, GO TO PASBAK
         C     R6,=F'1'                  ELSE ERCODE = 1 ?
         BE    ERRINV                    OPERAND INVALID
         B     ERLONG                    OPERAND LONG
*   PASS ONE AND TWO BYTE VALUES BACK
PASBAK   EQU   *
         L     R6,NUMVAL
         C     R9,=F'2'                  ITYPE = 2 ?
         BE    TWOBYT                    GO TO TWOBYT LOADING
         C     R6,=F'255'                VALUE > 255
         BH    ERRINV                    IF YES, OPERAND INVALUD
         ST    R6,0(R11,R10)             ELSE PASS BACK AS CODE
         LA    R11,4(R11)                INCREMENT INDEX
         LA    R2,1(R2)                  INCREMENT POSITION
         ST    R2,ADDROP                 STORE POSITION IN ADDROP
         C     R8,=F'1'                  LAST OP FLAG = 1 ?
         BE    DONE2                     IF YES, GO TO DONE2
         B     COMCNT                    CONTINUE
TWOBYT   EQU   *
         C     R6,=F'65536'              OVER TWO BYTES?
         BL    SPLIT                     IF LOW, GO TO SPLIT
```

```
         S     R6,=F'65536'              IF HIGH, SUBTRACT
         B     TWOBYT                    LOOP BACK TO TWOBYT
SPLIT    EQU   *
         ST    R6,NUMVAL                 STORE NUMVAL
         NI    NUMVAL+2,X'00'            ZERO UPPER BYTE
         L     R7,NUMVAL                 LOAD VALUE
         ST    R7,0(R11,R10)             PASS BACK AS CODE
         LA    R11,4(R11)                INCREMENT R11
         LA    R7,0                      ZERO R7
         ST    R7,NUMVAL                 ZERO NUMVAL
         ST    R6,SPACE                  STORE NUMVAL IN SPACE
         MVC   NUMVAL+3(1),SPACE+2       LOAD HIGH BYTE
         L     R6,NUMVAL                 LOAD BYTE FOR PASS
         ST    R6,0(R11,R10)             PASS BACK AS CODE
         LA    R11,4(R11)                INCREMENT R11
         LA    R2,1(R2)                  INCREMENT POSITION
         ST    R2,ADDROP                 STORE POSITION IN ADDROP
         C     R8,=F'1'                  CHECK LAST OP FLAG
         BE    DONE2                     IF SET, LEAVE ROUTINE
         B     COMCNT                    ELSE CONTINUE
INCR     EQU   *
         LA    R2,1(R2)                  INCREMENT POSITION
         B     COMCNT
*
*    COUNT NUMBER OF CHARACTERS IN QUOTES FOR STRINGS
*
QUOTES   EQU   *
         LA    R2,1(R2)                  SET POSITION
         LA    R4,0                      SET CHAR COUNT TO ZERO
         LA    R5,0                      ZERO R5
COUNT    EQU   *
         CLI   0(R2),C''''               CHAR '?
         BE    DONE2                     IF YES, END OF STRING
         LA    R4,1(R4)                  INCREMENT COUNT
         C     R4,LIMIT                  OVER LIMIT ?
         BH    ERLONG                    IF YES, GO TO ERLONG
         C     R3,=F'1'                  PASS = 1 ?
         BE    ADVANC                    GO TO ADVANC
         IC    R5,0(R2)                  ELSE LOAD CHAR IN R5
         ST    R5,0(R11,R10)             AND PASS BACK AS CODE
         LA    R11,4(R11)                INCREMENT R11
ADVANC   EQU   *
         LA    R2,1(R2)                  INCREMENT POSITION
         B     COUNT                     LOOP BACK TO COUNT
DONE     EQU   *
         C     R3,=F'2'                  PASS = 2 ?
         BNE   DONE2                     IF NOT, GO TO DONE2
         LA    R8,1                      ELSE LOAD LAST OP FLAG
         B     SUBOUT                    AND GO TO SUBOUT
DONE2    EQU   *
         L     R8,12(R1)                 LOAD ADDR NUMOPS
         ST    R4,0(R8)                  PASS BACK NUMOPS
DONE3    EQU   *
         B     LEAVE                     LEAVE ROUTINE
*    ERROR ROUTINES
ERRINV   EQU   *
         LA    R12,1                     SET ERROR CODE = 1
         B     LEAVE                     AND LEAVE ROUTINE
ERLONG   EQU   *
         LA    R12,2                     SET ERROR CODE = 2
```

```
LEAVE    EQU   *
         L     R8,16(R1)                  LOAD ADDR ERCODE
         ST    R12,0(R8)                  PASS BACK ERROR CODE
         $CLOSE                           IBM CLOSING CONVENTIONS
LIMIT    DS    1F
ADDRCP   DS    1F
NUMVAL   DS    1F
ERCODE   DS    1F
SPACE    DS    1F
         END   STRING
/*
//STEP9 EXEC ASMFC,PARM='NODECK,LOAD'
//ASM.SYSLIB DD DSN=IE170.MACLIB,DISP=SHR
//           DD DSN=SYS1.MACLIB,DISP=SHR
//ASM.SYSPRINT DD DUMMY
//ASM.SYSGO DD DSN=&&OBJECT,DISP=(MOD,PASS)
//ASM.SYSIN DD *
*********************************************************************
**********     ASSEMBLER SUBROUTINE,   TITLE : OPERAN     ***********
**********   CALL OPERAN (MOD,A(6),OPER1,IERCOD,OPER2,NVAL1,NVAL2) ***
*********************************************************************
OPERAN   CSECT
         $EQUR                            EQUATE REGISTERS
         $OPEN                            IBM OPENING CONVENTIONS
*
*   BEGIN PROCESSING BASED ON VALUE OF MOD
*
BEGIN    EQU   *
         LA    R12,0                      ZERO ERROR CODE
         ST    R12,ERCODE                 ZERO ERROR CODE FOR VALRED
         L     R2,0(R1)                   LOAD ADDR OF MOD
         L     R2,0(R2)                   LOAD MOD
         C     R2,=F'0'                   MOD = 0 ?
         BE    DATA                       IF YES, CC TO DATA
         LA    R11,1                      SET MODS PASS FLAG = 1
         L     R3,4(R1)                   LOAD ADDR OPERANDS
*
*   READ REGISTERS AS OPERANDS
*
PREREG   EQU   *
         MVC   REGIST(4),BLANKS           BLANK OUT REGIST AREA
         LA    R4,0                       ZERO INDEX
         LA    R5,0                       ZERO R5
         LA    R8,1                       SET INCREMENT
         LA    R9,3                       SET LIMIT
         CLI   0(R3),C' '                 FIRST CHAR BLANK?
         BE    ERRINV                     IF YES, CHAR INVALID
REGRED   EQU   *
         IC    R5,0(R4,R3)                INSERT CHAR INTO R5
         C     R5,=F'107'                 CHAR = , ?
         BE    FINISH                     IF SO, READ IS FINISHED
         C     R5,=F'64'                  CHAR = BLANK?
         BE    FINISH                     IF YES, DONE
         BXH   R4,R8,ERLONG               OVER 3 CHAR , TOO LONG
         STC   R5,REGIST-1(R4)            STORE IN REGIST
         B     REGRED                     GET NEXT CHAR
FINISH   EQU   *
         AR    R3,R4                      UPDATE OPERAND LOCATION
         LA    R3,1(R3)                   MOVE PAST DELIMITER
*
```

```
*    SELECT OPERAND LIST BY MOD TYPE
*
          C     R2,=F'2'                  MOD = 2 ?
          BH    PROC3                     IF NOT, GO TO PROC3
PROC12    EQU   *
          LA    R5,LIST1                  USE LIST1
          B     SEARCH                    GO TO SEARCH
PROC3     EQU   *
          C     R2,=F'3'                  MOD = 3 ?
          BH    PROC4                     IF NO, GO TO PROC4
          LA    R5,LIST2                  USE LIST2
          B     SEARCH                    GO TO SEARCH
PROC4     EQU   *
          C     R2,=F'4'                  MOD = 4 ?
          BH    PROC5                     IF NO, GO TO PROC5
          LA    R5,LIST3                  USE LIST3
PROC5     EQU   *
          LA    R5,LIST1                  USE LIST1
*
*    SEARCH OPERAND LIST AND MATCH TO FIND NUMBER VALUE OF REGISTER.
*    FAILURE TO MATCH INDICATES AN INVALID REGISTER
*
SEARCH    EQU   *
          CLC   REGIST(4),0(R5)           COMPARE REGIST WITH LIST
          BE    FOUND                     IF MATCH, GO TO FOUND
          CLI   0(R5),C'Z'                LIST ENTRY = Z ?
          BE    ERRINV                    MATCH NOT FOUND
          LA    R5,8(R5)                  INCREMENT ADDR
          B     SEARCH
FOUND     EQU   *
          LA    R5,4(R5)                  MOVE TO VALUE WORD
          L     R7,0(R5)                  LOAD VALUE
          C     R2,=F'5'                  MOD = 5 ?
          BE    MOD5                      IF YES, GO TO MOD5
*
*    SHIFT NUMBER THREE PLACES FOR MOD 2
*
          C     R2,=F'2'                  MOD = 2 ?
          BE    SHIFT                     SHIFT OPERAND VALUE
          B     PASSIT                    ELSE PASS AS IS
SHIFT     EQU   *
          LA    R6,0                      ZERO R6
          M     R6,=F'8'                  MULTIPLY TO GET 3 PLACE SHIFT
PASSIT    EQU   *
          L     R5,20(R1)                 LOAD ADDR NVAL1
          ST    R7,0(R5)                  PASS BACK NVAL1
          ST    R3,ADDROP                 STORE ADDROP
          C     R2,=F'5'                  MOD = 5 ?
          BE    MOD5                      IF YES, GO TO MOD5
          B     DATA2                     GO TO DATA2 FOR NEXT OPERANDS
*
*    FOR MOD 5, READ NEXT OPERAND AS REGISTER
*
MOD5      EQU   *
          LA    R11,1(R11)                INCREMENT MOD5 PASS FLAG
          C     R11,=F'2'                 MOD5 PASS FLAG = 2 ?
          BE    SHIFT                     IF YES, GO TO SHIFT
          C     R11,=F'3'                 MOD5 PASS FLAG = 3 ?
          BE    PREREG                    IF YES, GO TO PREREG
          L     R6,24(R1)                 ELSE LOAD ADDR NVAL2
```

```
         ST    R7,0(R5)                  PASS BACK NVAL2
         B     LEAVE                     AND LEAVE ROUTINE
*
*   ROUTINE TO READ LABEL AND NUMERICAL DATA
*
DATA     EQU   *
         L     R4,4(R1)                  LOAD ADDR OF POSITION
         ST    R4,ADDROP                 STORE ADDROP FOR CALL
         LA    R3,0                      FLAG REGISTER = 0
         LA    R6,20                     LOAD NVAL POINTER
         L     R2,8(R1)                  LOAD ADDR OPER1
         L     R2,0(R2)                  LOAD OPER1
         C     R2,=F'4'                  OPER1 = 4 ?
         BE    SUBOUT                    IF YES, GO TO SUBOUT
         LA    R3,1                      ELSE SET FLAG REGISTER
*
*   USE SUBROUTINE VALRED TO RETRIEVE NUMERICAL VALUES
*
SUBOUT   EQU   *
         LR    R5,R1                     SAVE R1 DURING CALL
         CALL  VALRED,(ADDROP,NUMVAL,ERCODE),VL   SUBR FOR NUMERICAL
         LR    R1,R5                     RESTORE R1
         L     R11,ERCODE                LOAD ERCODE
         C     R11,=F'0'                 ERCODE = 0 ?
         BE    OPRED                     IF YES, GO TO OPRED
         C     R11,=F'1'                 ERCODE = 1 ?
         BE    ERRINV                    IF YES, OPERAND INVALID
         B     ERLONG                    ELSE OPERAND TOO LONG
OPRED    EQU   *
         C     R3,=F'1'                  OPER1 = 5 ?
         BNE   NEXTOP                    IF NOT, GO TO NEXTOP
*
*   ROUTINE TO SPLIT LARGE NUMBERS INTO TWO BYTES, LOW ORDER FIRST
*
VALCUT   EQU   *
         L     R4,NUMVAL                 LOAD NUMVAL
VALTST   EQU   *
         C     R4,=F'65536'              NUMVAL OVER TWO BYTES?
         BL    CUTUP                     IF LESS, PROCEED WITH SPLIT
         S     R4,=F'65536'              ELSE MAKE MOD 65K
         B     VALTST                    LOOP BACK TO VALTST
CUTUP    EQU   *
         ST    R4,NUMVAL                 STORE NUMVAL
         NI    NUMVAL+2,X'00'            ZERO UPPER BYTE
         L     R5,NUMVAL                 LOAD NUMVAL
         L     R11,0(R6,R1)              LOAD PASS BACK ADDR
         ST    R5,0(R11)                 PASS BACK NVAL1
         LA    R6,4(R6)                  INCREMENT R6
         LA    R5,0                      ZERO R5
         ST    R5,NUMVAL                 ZERO NUMVAL
         ST    R4,SPACE                  LOAD NUMBER IN SPACE
         MVC   NUMVAL+3(1),SPACE+2       LOAD HIGH BYTE OF NUMBER
         L     R5,NUMVAL                 LOAD NUMVAL
         L     R11,0(R6,R1)              LOAD PASS BACK ADDR
         ST    R5,0(R11)                 PASS BACK NVAL2
         B     LEAVE                     AND LEAVE ROUTINE
*
*   LOAD SINGLE BYTE OPERANDS
*
NEXTOP   EQU   *
```

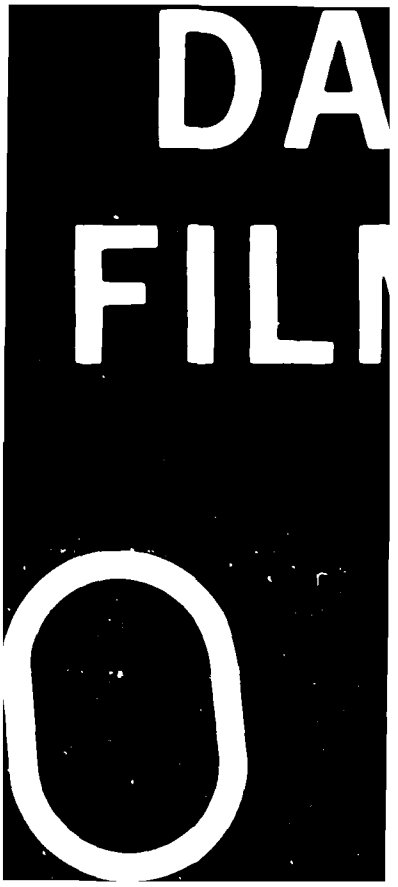
```
         C       R3,=F'3'                FLAG REGISTER = 3 ?
         BNE     TESTOP                  IF NOT, GO TO TESTOP
         LA      R6,24                   IF YES, LOAD NVAL POINTER
         B       VALOUT                  AND GO TO VALOUT
TESTOP   EQU     *
         L       R4,NUMVAL               LOAD NUMVAL
         C       R4,=F'255'              NUMVAL OVER ONE BYTE?
         BH      ERLONG                  IF YES, OPERAND IS TOO LONG
         C       R3,=F'2'                FLAG REGISTER = 2 ?
         BE      OPNUM2                  IF YES, GO TO OPNUM2
         L       R11,20(R1)              LOAD ADDR NVAL1
         ST      R4,0(R11)               PASS BACK NVAL1
*
*  PROCESS SECOND DATA OPERAND
*
DATA2    EQU     *
         L       R2,16(R1)               LOAD ADDR OPER2
         L       R2,0(R2)                LOAD OPER2
         C       R2,=F'0'                OPER2 = 0 ?
         BE      LEAVE                   IF YES, LEAVE ROUTINE
         LA      R3,2                    SET FLAG REGISTER TO 1
         C       R2,=F'4'                OPER2 = 4 ?
         BE      SUBOUT                  IF EQUAL, GO TO SUBOUT
         LA      R3,3                    SET FLAG REGISTER = 3
         B       SUBOUT                  GO TO SUBOUT
*
*   LOAD SECOND ONE BYTE OPERAND
*
OPNUM2   EQU     *
         L       R4,NUMVAL               LOAD NUMVAL
         L       R11,24(R1)              LOAD ADDR NVAL2
         ST      R4,0(R11)               PASS BACK NVAL2
         B       LEAVE                   AND LEAVE ROUTINE
*
*   ERROR HANDLING ROUTINES
*
ERRINV   EQU     *
         LA      R12,1                   SET ERROR CODE = 1
         B       LEAVE                   AND LEAVE ROUTINE
ERLONG   EQU     *
         LA      R12,2                   SET ERROR CODE = 2
LEAVE    EQU     *
         L       R11,12(R1)              LOAD IERCOD RETURN ADDR
         ST      R12,0(R11)              RETURN IERCOD
         $CLOSE                          RETURN
ADDROP   DS      1F
NUMVAL   DS      1F
ERCODE   DS      1F
SPACE    DS      1F
REGIST   DS      1F
BLANKS   DC      CL8'        '
*
*   REGISTER OPERAND LISTS
*
LIST1    EQU     *
         DC      CL4'A   '
         DC      F'7'
         DC      CL4'B   '
         DC      F'0'
         DC      CL4'C   '
```

```
          DC     F'1'
          DC     CL4'D   '
          DC     F'2'
          DC     CL4'E   '
          DC     F'3'
          DC     CL4'H   '
          DC     F'4'
          DC     CL4'L   '
          DC     F'5'
          DC     CL4'M   '
          DC     F'6'
          DC     CL4'Z   '
LIST2     EQU    *
          DC     CL4'B   '
          DC     F'0'
          DC     CL4'D   '
          DC     F'16'
          DC     CL4'H   '
          DC     F'32'
          DC     CL4'SP  '
          DC     F'48'
          DC     CL4'PSW '
          DC     F'48'
          DC     CL4'Z   '
LIST3     EQU    *
          DC     CL4'B   '
          DC     CL4'D   '
          DC     F'0'
          DC     F'16'
          DC     CL4'Z   '
          END OPERAN
/*
//STEP10 EXEC ASMFC,PARM='NODECK,LOAD'
//ASM.SYSLIB DD DSN=IE170.MACLIB,DISP=SHR
//           DD DSN=SYS1.MACLIB,DISP=SHR
//ASM.SYSPRINT DD DUMMY
//ASM.SYSGO DD DSN=&&OBJECT,DISP=(MOD,PASS)
//ASM.SYSIN DD *
*********************************************************************
**********      ASSEMBLER SUBROUTINE,    TITLE : VALRED      **********
**********           CALL VALRED (ADDROP,NUMVAL,ERCODE)      **********
*********************************************************************
VALRED    CSECT
          EXTRN LABOUT
          $EQUR                          EQUATE REGISTERS
          $OPEN                          IBM OPENING CONVENTIONS
BEGIN     EQU    *
          LA     R12,0                   ZERO ERROR CODE
          ST     R12,NUMVAL              ZERO NUMVAL
          ST     R12,ERCODE              ZERO EXTERNAL ERCODE
          ST     R12,EXCODE              ZERO EXCODE
          ST     R12,EXVAL               ZERO EXVAL
          L      R2,0(R1)                LOAD ADDR POSITION OF OPERANDS
          L      R4,8(R1)                LOAD ADDR ERCODE
          L      R4,0(R4)                LOAD INCOMING ERCODE
          C      R4,=F'0'                CHECK FOR OP ADDR MODE
          BNE    LBLTST                  IF NOT ZERO, LOAD ONLY ADDR OP
          L      R2,0(R2)                LOAD POSITION OF OPERANDS
LBLIST    EQU    *
          LA     R10,0                   ZERO EXPRESSION CODE
```

```
         CLI   0(R2),C'A'          FIRST CHAR = A ?
         BL    ERRINV              IF LOW, INVALID OPERAND
         CLI   0(R2),C'Z'          CHAR = Z ?
         BH    NUMTST              IF HIGH, TEST FOR NUMBER
         LA    R9,6                ELSE LOAD MAX CHAR = 6
         B     SETUP               AND BRANCH TO SETUP
NUMTST   EQU   *
         CLI   0(R2),C'0'          CHAR = 0 ?
         BL    ERRINV              IF LOW INVALID OPERAND
         CLI   0(R2),C'9'          CHAR = 9 ?
         BH    ERRINV              IF HIGH, INVALID OPERAND
         LA    R9,9                ELSE LOAD MAX CHAR = 9
SETUP    EQU   *
         MVC   LABNUM(9),BLANKS    FILL SPACE WITH BLANKS
         LA    R8,1                SET INCREMENT = 1
         LA    R5,0                ZERO R5
         LA    R4,0                ZERO INDEX
INDATA   EQU   *
         IC    R5,0(R2)            GET CHARACTER
         C     R5,=F'64'           CHAR = BLANK
         BE    DONE                IF YES, DONE
         C     R5,=F'107'          CHAR = , ?
         BE    DONE                IF YES, DONE
         C     R5,=F'78'           CHAR = + ?
         BE    XPRES1              IF YES, GO TO XPRES1
         C     R5,=F'96'           CHAR = - ?
         BE    XPRES2              IF YES, GO TO XPRES2
         BXH   R4,R8,ERLONG        IF CHAR OVER MAX, GO TO ERLONG
         STC   R5,LABNUM-1(R4)     STORE CHAR IN LABNUM
         LA    R2,1(R2)            INCREMENT POSITION
         B     INDATA              GET NEXT CHAR
XPRES1   EQU   *
         LA    R10,1               SET EXPRESSION CODE = 1
         B     DONE                GO TO DONE
XPRES2   EQU   *
         LA    R10,2               SET EXPRESSION CODE = 2
DONE     EQU   *
         C     R9,=F'6'            DATA IS LABEL ?
         BH    NUMBER              IF NO, GO TO NUMBER
         LR    R5,R1               SAVE R1 DURING CALL
         CALL  LABOUT,(LABNUM,NUMVAL,ERCODE),VL
         LR    R1,R5               RESTORE R1
         L     R3,ERCODE           LOAD ERROR CODE
         C     R3,=F'1'            ERCODE = 1 ?
         BE    ERRINV              INVALID OR NOT FOUND VLAUE
VALFND   EQU   *
         LA    R2,1(R2)            MOVE POINTER TO NEXT OPERAND
         L     R7,EXCODE           LOAD EXPRES CODE
         C     R7,=F'0'            CODE = 0 ?
         BE    EXPTST              IF YES, GO TO EXPTST
         C     R7,=F'1'            CODE = 1 ?
         BNE   SUBEXP              IF NOT, GO TO SUBTRACT
ADDEXP   EQU   *
         L     R7,EXVAL            LOAD EXPRESSION VALUE
         A     R7,NUMVAL           ADD NUMVAL
         ST    R7,NUMVAL           STORE RESULT IN NUMVAL
         B     EXPTST              GO TO EXPTST
SUBEXP   EQU   *
         L     R7,EXVAL            LOAD EXVAL
         S     R7,NUMVAL           SUBTRACT NUMVAL
```

```
         ST    R7,NUMVAL                STORE RESULT IN NUMVAL
EXPTST   EQU   *
         C     R10,=F'0'                EXPRESSION CODE = 0 ?
         BE    VALOUT                   IF YES, GO TO VALOUT
         L     R7,NUMVAL                ELSE, LOAD NUMVAL
         ST    R7,EXVAL                 STORE IN EXVAL
         ST    R10,EXCODE               STORE EXPRESS CODE IN EXCODE
         B     LBLTST                   GO TO LBLTST
VALOUT   EQU   *
         L     R6,NUMVAL                LOAD NUMVAL
         L     R7,4(R1)                 LOAD ADDR NUMVAL
         ST    R6,0(R7)                 PASS BACK NUMVAL
         B     LEAVE                    AND LEAVE ROUTINE
NUMBER   EQU   *
         S     R2,=F'1'                 POINT R2 TO LAST VALID CHAR
         CLI   0(R2),C'H'               LAST CHAR = H ?
         BE    HEXIN                    IF YES, GO TO HEXIN
         CLI   0(R2),C'O'               LAST CHAR = O ?
         BE    OCTIN                    IF YES, GO TO OCTIN
         CLI   0(R2),C'B'               LAST CHAR = B ?
         BE    BININ                    IF YES, GO TO BININ
DECIN    EQU   *
         LA    R7,10                    LOAD MULTIPLIER
         B     CONVRT
OCTIN    EQU   *
         LA    R7,8                     LOAD MULTIPLIER
         S     R4,=F'1'                 POINT LIMIT AT LAST DIGIT
         B     CONVRT                   GO TO CONVRT
BININ    EQU   *
         LA    R7,2                     LOAD MULTIPLIER
         S     R4,=F'1'                 POINT LIMIT AT LAST DIGIT
CONVRT   EQU   *
         LA    R9,0                     ZERO VALUE
         LA    R5,0                     ZERO POINTER INDEX
         LA    R8,0                     ZERO R8
         LA    R6,15                    LOAD MASK FOR CHAR TO BINARY
CTLOOP   EQU   *
         MR    R8,R7                    MULTIPLY BY BASE
         IC    R3,LABNUM(R5)            LOAD NEXT CHAR
         NR    R3,R6                    CONVERT TO BINARY
         AR    R9,R3                    ADD TO SUM
         LA    R5,1(R5)                 INCREMENT POINTER
         BCT   R4,CTLOOP                DO LOOP BY NUMBER OF DIGITS
         ST    R9,NUMVAL                STORE VALUE IN NUMVAL
         LA    R2,1(R2)                 RESTORE VALUE OF R2
         B     VALEND                   GO TO VALEND
HEXIN    EQU   *
         LA    R7,16                    LOAD MULTIPLIER
         S     R4,=F'1'                 POINT LIMIT AT LAST DIGIT
         LA    R5,0                     ZERO INDEX POINTER
         LA    R8,0                     ZERO R8
         LA    R9,0                     ZERO VALUE
         ST    R9,CHARIN                ZERO CHARIN
HEXCVT   EQU   *
         MR    R8,R7                    MULTIPLY BY BASE
         LA    R3,0                     ZERO LIST INDEX
         LA    R11,0                    ZERO R11
         LA    R6,16                    SET LOOP COUNTER FOR LIST
         IC    R11,LABNUM(R5)           LOAD CHAR
         ST    R11,CHARIN               STORE CHAR
```

```
IDLOOP   EQU   *
         IC    R11,HXLIST(R3)          INSERT CHAR FROM LIST
         C     R11,CHARIN              COMPARE CHAR AND LIST ENTRY
         BE    HEXVAL                  IF EQUAL, GO TO HEXVAL
         LA    R3,1(R3)                INCREMENT R3
         BCT   R6,IDLOOP               GO BACK TO IDLOOP ON COUNT
         B     ERRINV                  IF NOT FOUND, INVALID CHAR
HEXVAL   EQU   *
         AR    R9,R3                   ADD TO SUM
         LA    R5,1(R5)                INCREMENT POINTER INDEX
         BCT   R4,HEXCVT               BRANCH ON NUMBER OF DIGITS
         ST    R9,NUMVAL               STORE VALUE IN NUMVAL
         LA    R2,1(R2)                RESTORE VALUE OF R2
         B     VALEND                  GO TO VALEND
ERRINV   EQU   *
         LA    R12,1                   SET ERCODE = 1
         B     LEAVE                   AND LEAVE ROUTINE
ERLONG   EQU   *
         LA    R12,2                   SET ERCODE = 2
LEAVE    EQU   *
         L     R11,8(R1)               LOAD ADDR OF ERCODE
         ST    R12,0(R11)              PASS BACK ERCODE
END      EQU   *
         $CLOSE
LABNUM   DS    3F
NUMVAL   DS    1F
ERCODE   DS    1F
EXVAL    DS    1F
EXCODE   DS    1F
CHARIN   DS    1F
HXLIST   DC    CL16'0123456789ABCDEF'
BLANKS   DC    CL16'                '
         END   VALREQ
```


DA
FIL